

# Electronic Notes in Theoretical Computer Science

---

---

THE THIRTY-SECOND CONFERENCE ON THE MATHEMATICAL FOUNDATIONS OF PROGRAMMING SEMANTICS

**MFPS XXXII**

Pittsburgh, USA

May 23-26, 2016

---

Guest Editor:

LARS BIRKEDAL

---



# Contents

<b>Preface</b>	v
DAVID MESTEL AND BILL ROSCOE Reducing complex CSP models to traces via priority .....	1
RICHARD STATMAN On the representation of semigroups and other congruences in the lambda calculus .....	16
FABIO ZANASI The algebra of partial equivalence relations .....	24
RICHARD STATMAN How to think of intersection types as Cartesian products .....	45
TETSUYA SATO Approximate Relational Hoare Logic for Continuous Random Samplings .....	55
JURRIAAN ROT Coalgebraic minimization of automata by initiality and finality .....	70
BART JACOBS AND FABIO ZANASI A predicate/state transformer semantics for Bayesian learning .....	94
TOMÁŠ JAKL, ACHIM JUNG AND ALEŠ PULTR Bitopology and four-valued logic .....	109
BART JACOBS Effectuses from Monads .....	127
BARRY JAY Programs as Data Structures in $\lambda SF$ -Calculus .....	141
TYLER BARKER A Monad for Randomized Algorithms .....	156
ARTHUR AZEVEDO DE AMORIM Binding Operators for Nominal Sets .....	171
BRAM GERON AND PAUL BLAIN LEVY Iteration and labelled iteration .....	195
FREDRIK DAHLQVIST, VINCENT DANOS AND ILIAS GARNIER Giry and the Machine .....	213

SERGEY GONCHAROV, STEFAN MILIUS AND CHRISTOPH RAUCH	
Complete Elgot Monads and Coalgebraic Resumptions .....	238
JONAS FREY	
Classical realizability in the CPS target language .....	259
ROBIN COCKETT AND JONATHAN GALLAGHER	
Categorical Models of the Differential $\lambda$ -Calculus Revisited .....	274
MARC BAGNOL, RICHARD BLUTE, J.R.B. COCKETT AND J.S. LEMAY	
The shuffle quasimonad and modules with differentiation and integra- tion .....	295

## Preface

This volume contains the proceedings of the The Thirty-second Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXII). The conference is held in Pittsburgh, USA, between May 23 and May 26th. MFPS conferences are devoted to those areas of mathematics, logic, and computer science that are related to models of computation, in general, and to the semantics of programming languages, in particular. The series has particularly stressed providing a forum where researchers in mathematics and computer science can meet and exchange ideas about problems of common interest. As the series also strives to maintain breadth in its scope, the conference strongly encourages participation by researchers in neighbouring areas. The program committee of MFPS XXXII consisted of

- Achim Jung, Birmingham, UK
- Andre Scedrov, UPenn, USA
- Andrej Bauer, Ljubljana, Slovenia
- Andrzej Murawski, Warwick, UK,
- Bart Jacobs, Radboud U, Netherlands
- Bob Coecke, Oxford, UK
- Cameron Freer, Cambridge MA, USA
- Catherine Meadows, NRL , USA
- Catuscia Palamidessi, INRIA, France
- Christine Tasson, PPS Paris, France
- Claudio Russo, MSR Cambridge, UK
- Dusko Pavlovic, Hawaii, US
- Helle Hvid Hansen, TU Delft, Netherlands
- Hugo Herbelin, Paris, France
- Jean Krivine, Paris, France,
- Joel Ouaknine, Oxford, UK
- Lars Birkedal (Chair), Aarhus, Denmark
- Michael Mislove, Tulane, USA
- Neel Krishnaswami, Birmingham, UK
- Paul Blain Levy, Birmingham, UK,

- Peter Dybjer, Chalmers, Sweden
- Prakash Panangaden, Montreal, Canada
- Stefan Milius, Erlangen, Germany
- Steve Brookes, CMU, USA
- Steve Zdancewic, UPenn, USA

The papers were refereed by the program committee and by several outside referees, whose help is gratefully acknowledged. The invited speakers at the conference were

- Peter Selinger, Dalhousie, Canada
- Brigitte Pientka, McGill, Canada
- Steve Brookes, CMU, USA
- Nathalie Bertrand, Inria, France

The conference also included four special sessions, one celebrating the 60th Birthday of Steve Brookes, and three featuring invited tutorials given by the special session organisers:

- Concurrency, special session in honour of Steve Brookes' 60th Birthday, organized by Peter O'Hearn (Facebook, UK) which included talks by Sir Tony Hoare (Microsoft, UK), Bill Roscoe (Oxford, UK), Viktor Vafeiadis (MPI, Germany).
- Verification, organized by Andrew Appel (Princeton, USA), which included talks by Steve Zdancewic (UPenn, USA), Gordon Stewart (Ohio, USA), Jan Hoffmann (CMU, USA).
- Security, organized by Stephen Chong (Harvard, USA) which included talks by Aslan Askarov (Aarhus, Denmark), Andrew Myers (Cornell, USA), Geoffrey Smith (Florida, USA).
- Probabilistic Programming, organized by Dan Roy (Toronto, Canada) which included talks by Johannes Borgstrom (Uppsala, Sweden), Sam Staton (Oxford, UK), Ken Shan (Indiana, USA).

MFPS gratefully acknowledges the financial support of the U.S. Office of Naval Research, and Carnegie Mellon University, USA. We also thank the local organizer Steve Brookes and his team for their organization of the conference.

*Aarhus, May 3, 2016*

*Lars Birkedal*

# Reducing complex CSP models to traces via priority

David Mestel<sup>1</sup> A.W. Roscoe<sup>2</sup>

*Department of Computer Science, University of Oxford*

---

## Abstract

Hoare's Communicating Sequential Processes (CSP) [6] admits a rich universe of semantic models. In this paper we study finite observational models, of which at least six have been identified for CSP, namely traces, failures, revivals, acceptances, refusal testing and finite linear observations [11]. We show how to use the recently-introduced *priority* operator ([12], ch.20) to transform refinement questions in these models into trace refinement (language inclusion) tests. Furthermore, we are able to generalise this to any (rational) finite observational model. As well as being of theoretical interest, this is of practical significance since the state-of-the-art refinement checking tool FDR3 [4] currently only supports two such models.

*Keywords:* CSP, denotational semantics, priority

---

## 1 Introduction

A number of different forms of process calculus have been developed for the modelling of concurrent programs, including Hoare's Communicating Sequential Processes (CSP) [6], Milner's Calculus of Communicating Systems (CCS) [7], and the  $\pi$ -calculus [8]. Unlike the latter two, CSP's semantics are traditionally given in behavioural semantic models coarser than bisimulation.

In this paper, we study finite linear-time observational models for CSP; that is, models where all observations considered can be determined in a finite time by an experimenter who can see the visible events a process communicates and the sets of events it can offer in any stable state. While the experimenter can run the process arbitrarily often, he or she can only record the results of individual finite executions. Thus each behaviour recorded can be deduced from a single finite sequence of events together with the sets of events accepted in stable states during and immediately after this *trace*.

At least six such models have been considered for CSP, but the state-of-the-art refinement checking tool, FDR3 [4], currently only supports two, namely *traces*

---

<sup>1</sup> Email: [david.mestel@cs.ox.ac.uk](mailto:david.mestel@cs.ox.ac.uk)

<sup>2</sup> Email: [bill.roscoe@cs.ox.ac.uk](mailto:bill.roscoe@cs.ox.ac.uk)

and *failures* (it also supports the failures-divergences model, which is not finite observational).

We present a construction which produces a context  $\mathcal{C}$  such that refinement questions in the failures model correspond to trace refinement questions under the application of  $\mathcal{C}$ . We are able to generalise this to show (Theorem 5.4) that a similar construction is possible not only for the six models which have been studied, but also for any sensible finite observational model (where ‘sensible’ means that the model can be recognised by a finite-memory computer, in a sense which we shall make precise).

We first briefly describe the language of CSP. We next give an informal description of our construction for the failures model. To prove the result in full generality, we first give a formal definition of a finite observational model, and of the notion of rationality. We then describe our general construction. Finally we discuss performance and optimisation issues.

## 2 The CSP language

We provide a brief outline of the language, largely taken from [11]; the reader is encouraged to consult [12] for a more comprehensive treatment.

Throughout,  $\Sigma$  is taken to be a finite nonempty set of communications that are visible and can only happen when the observing environment permits via handshaken communication. The actions of every process are taken from  $\Sigma \cup \{\tau\}$ , where  $\tau$  is the invisible internal action that cannot be prevented by the environment. Note that the usual treatment of CSP permits sequential composition by including another un-preventable event  $\checkmark$  to represent termination; this adds slight complications to each model and we omit it for simplicity. It could be added back without any significant alteration to the results of this paper.

The constant processes of CSP are

- *STOP* which does nothing—a representation of deadlock.
- *div* which performs (only) an infinite sequence of internal  $\tau$  actions—a representation of divergence or livelock.
- *CHAOS* which can do anything except diverge.

The prefixing operator introduces communication:

- $a \rightarrow P$  communicates the event  $a$  before behaving like  $P$ .

There are two forms of binary choice between a pair of processes:

- $P \sqcap Q$  lets the process decide to behave like  $P$  or like  $Q$ : this is *nondeterministic* or *internal* choice.
- $P \square Q$  offers the environment the choice between the initial  $\Sigma$ -events of  $P$  and  $Q$ . If the one selected is unambiguous then it continues to behave like the one chosen; if it is an initial event of both then the subsequent behaviour is nondeterministic. The occurrence of  $\tau$  in one of  $P$  and  $Q$  does *not* resolve the choice (unlike CCS +). This is *external* choice.

We only have a single parallel operator in our core language since all the usual



ones of CSP can be defined in terms of it as discussed in Chapter 2 etc. of [12].

- $P \parallel_X Q$  runs  $P$  and  $Q$  in parallel, allowing each of them to perform any action in  $\Sigma \setminus X$  independently, whereas actions in  $X$  must be synchronised between the two.

There are two operators that change the nature of a process's communications.

- $P \setminus X$ , for  $X \subseteq \Sigma$ , *hides*  $X$  by turning all  $P$ 's  $X$ -actions into  $\tau$ s.
- $P[[R]]$  applies the *renaming* relation  $R \subseteq \Sigma \times \Sigma$  to  $P$ : if  $(a, b) \in R$  and  $P$  can perform  $a$ , then  $P[[R]]$  can perform  $b$ . The domain of  $R$  must include all visible events used by  $P$ . Renaming by the relation  $\{(a, b)\}$  is denoted  $[[a/b]]$ .

There is another operator that allows one process to follow another:

- $P\Theta_A Q$  behaves like  $P$  until an event in the set  $A$  occurs, at which point  $P$  is shut down and  $Q$  is started. This is the *throw* operator.

The final CSP construct is *recursion*: this can be single or mutual (including mutual recursions over infinite parameter spaces), can be defined by systems of equations or (in the case of single recursion) in line via the notation  $\mu p.P$ , for a term  $P$  that may include the free process identifier  $p$ . Recursion can be interpreted operationally as having a  $\tau$ -action corresponding to a single unwinding. Denotationally, we regard  $P$  as a function on the space of denotations, and interpret  $\mu p.P$  as the least fixed point of this function.

We also make use of the *interleaving* operator  $|||$ , which allows processes to perform actions independently and is equivalent to  $\parallel_{\emptyset}$ , and the process  $RUN_X$ , which always offers every element of the set  $X$  and is defined by  $RUN_X = \bigsqcup_{x \in X} x \rightarrow RUN_X$ .

## 2.1 Priority

The prioritisation operator is discussed in detail in Chapter 20 of [12]. It allows us to specify an ordering on the set of visible events  $\Sigma$ , and prevents lower-priority events from occurring whenever a higher-priority event or  $\tau$  is available.

The operator described in [12] as implemented in FDR3 [4] is parametrised by three arguments: a process  $P$ , a partial order  $\leq$  on the event set  $\Sigma$ , and a subset  $X \subseteq \Sigma$  of events that can occur when a  $\tau$  is available. We require that all elements of  $X$  are maximal with respect to  $\leq$ . Writing  $initials(P) \subseteq \Sigma \cup \{\tau\}$  for the set of events that  $P$  can immediately perform, and extending  $\leq$  to a partial order on  $\Sigma \cup \{\tau\}$  by adding  $y \leq \tau \forall y \in \Sigma \setminus X$ , we define the operational semantics of prioritise as follows:

$$\frac{P \xrightarrow{a} P' \wedge \forall b \neq a. a \leq b \Rightarrow b \notin initials(P)}{prioritise(P, \leq, X) \xrightarrow{a} prioritise(P', \leq, X)} \quad (a \in \Sigma \cup \{\tau\}).$$

Note that prioritise is not compositional over denotational models other than the most precise model  $\mathcal{FL}$ , so we think of it as an optional addition to CSP rather than an integral part of it; when we refer below to particular types of observation as giving rise to valid models for CSP, we will mean CSP without priority.

### 3 Example: the failures model

We first demonstrate our construction using the *failures* model: we will produce a context  $\mathcal{C}$  such that for any processes  $P, Q$ , we have that  $Q$  refines  $P$  in the failures model if and only if  $\mathcal{C}[Q]$  refines  $\mathcal{C}[P]$  in the traces model.

#### 3.1 The traces and failures models

The *traces* model  $\mathcal{T}$  is familiar from automata theory, and represents a process by the set of (finite) strings of events it is able to accept. Thus each process is associated (for fixed alphabet  $\Sigma$ ) to a subset of  $\Sigma^*$  the set of finite words over  $\Sigma$ .

The *failures* model  $\mathcal{F}$  also records sets  $X$  of events that the process is able to stably refuse after a trace  $s$  (that is, the process is able after trace  $s$  to be in a state where no  $\tau$  events are possible, and where the set of initial events does not meet  $X$ ). Thus a process is associated to a subset of  $\Sigma^* \times (\mathcal{P}(\Sigma) \cup \{\bullet\})$ , where  $\bullet$  represents the absence of a recorded refusal set.<sup>3</sup> Note that recording  $\bullet$  does not imply that there is no refusal to observe, simply that we have not observed stability. The observation of the refusal  $\emptyset$  implies that the process can be stable after the present trace, whereas  $\bullet$  does not.

In any model  $\mathcal{M}$ , we say that  $Q$   $\mathcal{M}$ -refines  $P$ , and write  $P \sqsubseteq_{\mathcal{M}} Q$ , if the set associated to  $Q$  is a subset of that corresponding to  $P$ .

#### 3.2 Model shifting for the failures model

The construction is as follows:

**Lemma 3.1** *For each finite alphabet  $\Sigma$  there exists a context  $\mathcal{C}$  (over an expanded alphabet) such that for any processes  $P$  and  $Q$  we have that  $P \sqsubseteq_F Q$  if and only if  $\mathcal{C}[P] \sqsubseteq_T \mathcal{C}[Q]$ .*

**Proof. Step 1:** We use priority to produce a process (over an expanded alphabet) that can communicate an event  $x'$  if and only if the original process  $P$  is able to stably refuse  $x$ .

This is done by expanding the alphabet  $\Sigma$  to  $\Sigma \cup \Sigma'$  (where  $\Sigma'$  contains a corresponding primed event for every event in  $\Sigma$ ), and prioritising with respect to a partial order which prioritises each  $x$  over the corresponding  $x'$ . Recall that the definition of the priority operator means that this also causes  $\tau$  to be promoted over the primed events.

We must also introduce an event *stab* to signify stability without requiring any refusals to be possible. This is necessary in order to be able to record an empty refusal set. Let the partial order  $\leq_1$  be defined by  $x' <_1 x \forall x \in \Sigma$ , and let the context  $\mathcal{C}_1$  be defined by

$$\mathcal{C}_1[P] = \text{prioritise}(P \parallel \text{RUN}_{\Sigma' \cup \{\text{stab}\}}, \leq_1, \Sigma).$$

This process has a state  $\xi'$  for each state  $\xi$  of  $P$ , where  $\xi'$  has the same unprimed events (and corresponding transitions) as  $\xi$ . Furthermore  $\xi'$  can communicate  $x'$  just when  $\xi$  is stable and can refuse  $X$ , and *stab* just when  $\xi$  is stable.

<sup>3</sup> This is equivalent to the standard presentation in which a process is represented by a subset of  $\Sigma^*$  and one of  $\Sigma^* \times \mathcal{P}(\Sigma)$ : the trace component is just  $\{s : (s, \bullet) \in \mathcal{P}\}$ .

**Step 2:** We now recall that the definition of the failures model only allows a refusal set to be recorded at the *end* of a trace, and is not interested in (so does not record) what happens after the refusal set.

We gain this effect by using a regulator process to prevent a primed event (or *stab*) from being followed by an unprimed event. Let

$$\begin{aligned} UNSTABLE &= \Box_{x \in \Sigma} x \rightarrow UNSTABLE \\ &\quad \Box \Box_{x \in \Sigma' \cup \{stab\}} x \rightarrow STABLE \\ STABLE &= \Box_{x \in \Sigma' \cup \{stab\}} x \rightarrow STABLE, \end{aligned}$$

and define  $\mathcal{C}$  by

$$\mathcal{C}[P] = \mathcal{C}_1[P] \parallel_{\Sigma \cup \Sigma' \cup \{stab\}} UNSTABLE.$$

A trace of  $\mathcal{C}[P]$  consists of: firstly, a trace  $s$  of  $P$ ; followed by, if  $P$  can after  $s$  be in a stable state, then for some such state  $\sigma_0$  any string formed from the events that can be refused in  $\sigma_0$ , together with *stab*. The lemma clearly follows.  $\square$

It is clear that any such context must involve an operator that is not compositional over traces, for otherwise we would have  $P \sqsubseteq_T Q$  implies  $\mathcal{C}[P] \sqsubseteq_T \mathcal{C}[Q]$ , which is equivalent to  $P \sqsubseteq_F Q$ , and this is not true for general  $P$  and  $Q$  (consider for instance  $P = a \rightarrow STOP$ ,  $Q = (a \rightarrow STOP) \sqcap STOP$ ). It follows that only contexts which like ours involve priority can achieve this.

## 4 Semantic models

In order to generalise this construction to arbitrary finite observational semantic models, we must give formal definitions not only of particular models but of the very notion of a finite observational model.

### 4.1 Finite observations

We consider only models arising from *finite linear observations*. Intuitively, we postulate that we are able to observe the process performing a finite number of visible actions, and that where the process was stable (unable to perform a  $\tau$ ) immediately before an action, we are able to observe the *acceptance set* of actions it was willing to perform.

Note that we are unable to finitely observe *instability*: the most we are able to record from an action in an unstable state is that we did not *observe* stability. Thus in any context where we can observe stability we can also fail to observe it by simply not looking.

We take models to be defined over finite alphabets  $\Sigma$ , and take an arbitrary ordering on each finite  $\Sigma$  to be *alphabetical*.

The most precise finite observational model is that considering all finite linear observations, and is denoted  $\mathcal{FL}$ :

**Definition 4.1** The set of *finite linear observations* over an alphabet  $\Sigma$  is

$$\mathcal{FL}_\Sigma := \{\langle A_0, a_1, A_1, \dots, A_{n-1}, a_n, A_n \rangle : n \in \mathbb{N}, a_i \in \Sigma, A_i \subseteq \Sigma \text{ or } A_i = \bullet\},$$

where the  $a_i$  are interpreted as a sequence of communicated events, and the  $A_i$  denote stable acceptance sets, or in the case of  $\bullet$  failure to observe stability. Let the set of such observations corresponding to a process  $P$  be denoted  $\mathcal{FL}_\Sigma(P)$ .

(Sometimes we will drop the  $\Sigma$  and just write  $\mathcal{FL}(P)$ ).

More formally,  $\mathcal{FL}(P)$  can be defined inductively; for instance

$$\mathcal{FL}(P \sqcap Q) := \{\langle A \cup B \rangle^\alpha, \langle A \cup B \rangle^\beta : \langle A \rangle^\alpha \in \mathcal{FL}(P), \langle B \rangle^\beta \in \mathcal{FL}(Q)\}$$

(where  $X \cup \bullet := \bullet$  for any set  $X$ ). See Section 11.1.1 of [12] for further details.

Observe that  $\mathcal{FL}$  has a natural partial order corresponding to prefixes (where  $\alpha \hat{\langle \bullet \rangle}^\beta$  and  $\alpha \hat{\langle A \rangle}^\beta$  are both considered prefixes of  $\alpha \hat{\langle A \rangle}^\beta$  for any set  $A$  and any  $\alpha$  and  $\beta$ ). Note that for any process  $P$  we have that  $\mathcal{FL}(P)$  is downwards-closed with respect to this partial order. This can be interpreted as saying that  $\alpha \leq \beta$  if the presence of the observation  $\beta$  implies that  $\alpha$  is an observation that could have been made of the same experiment on  $P$ .

#### 4.2 Finite observational models

We consider precisely the models which are derivable from the observations of  $\mathcal{FL}$ , which are well-defined in the sense that they are compositional over CSP syntax (other than priority), and which respect extension of the alphabet  $\Sigma$ .

**Definition 4.2** A finite observational *pre-model*  $\mathcal{M}$  consists for each (finite) alphabet  $\Sigma$  of a set of *observations*,  $\text{obs}_\Sigma(\mathcal{M})$ , together with a relation  $\mathcal{M}_\Sigma \subseteq \mathcal{FL}(\Sigma) \times \text{obs}_\Sigma(\mathcal{M})$ . The representation of a process  $P$  in  $\mathcal{M}_\Sigma$  is denoted  $\mathcal{M}_\Sigma(P)$ , and is given by

$$\mathcal{M}_\Sigma(P) := \mathcal{M}_\Sigma(\mathcal{FL}_\Sigma(P)) = \{y \in \text{obs}_\Sigma(\mathcal{M}) : \exists x \in \mathcal{FL}_\Sigma(P). (x, y) \in \mathcal{M}_\Sigma\}.$$

For processes  $P$  and  $Q$  over alphabet  $\Sigma$ , if we have  $\mathcal{M}_\Sigma(Q) \subseteq \mathcal{M}_\Sigma(P)$  then we say  $Q$   *$\mathcal{M}$ -refines*  $P$ , and write  $P \sqsubseteq_{\mathcal{M}} Q$ .

(As before we will sometimes drop the  $\Sigma$ ).

Note that this definition is less general than if we had defined a pre-model to be any equivalence relation on  $\mathcal{P}(\mathcal{FL}_\Sigma)$ . For example, the equivalence relating sets of the same cardinality has no corresponding pre-model. Definition 4.2 agrees with that sketched in [12].

Without loss of generality,  $\mathcal{M}_\Sigma$  does not identify any elements of  $\text{obs}_\Sigma(\mathcal{M})$ ; that is, we have  $\mathcal{M}_\Sigma^{-1}(x) = \mathcal{M}_\Sigma^{-1}(y)$  only if  $x = y$  (otherwise quotient by this equivalence relation). Subject to this assumption,  $\mathcal{M}_\Sigma$  induces a partial order on  $\text{obs}_\Sigma(\mathcal{M})$ :

**Definition 4.3** The partial order *induced by  $\mathcal{M}_\Sigma$  on  $\text{obs}_\Sigma(\mathcal{M})$*  is given by:  $x \leq y$  if and only if for all  $b \in \mathcal{M}_\Sigma^{-1}(y)$  there exists  $a \in \mathcal{M}_\Sigma^{-1}(x)$  with  $a \leq b$ .

Observe that for any process  $P$  it follows from this definition that  $\mathcal{M}(P)$  is downwards-closed with respect to this partial order (since  $\mathcal{FL}(P)$  is downwards-closed).

**Definition 4.4** A pre-model  $\mathcal{M}$  is *compositional* if for all CSP operators  $\oplus$ , say of arity  $k$ , and for all processes  $P_1, \dots, P_k$  and  $Q_1, \dots, Q_k$  such that  $\mathcal{M}(P_i) = \mathcal{M}(Q_i)$  for all  $i$ , we have

$$\mathcal{M}\left(\bigoplus(P_i)_{i=1\dots k}\right) = \mathcal{M}\left(\bigoplus(Q_i)_{i=1\dots k}\right).$$

This means that the operator defined on processes in  $\text{obs}(\mathcal{M})$  by taking the pushforward of  $\oplus$  along  $\mathcal{M}$  is well-defined: for any sets  $X_1, \dots, X_k \subseteq \text{obs}(\mathcal{M})$  which correspond to the images of CSP processes, take processes  $P_1, \dots, P_k$  such that  $X_i = \mathcal{M}(P_i)$ , and let

$$\bigoplus(X_i)_{i=1\dots k} = \mathcal{M}\left(\bigoplus(P_i)_{i=1\dots k}\right).$$

Definition 4.4 says that the result of this does not depend on the choice of the  $P_i$ .

Note that it is not necessary to require the equivalent of Definition 4.4 for recursion in the definition of a model, because of the following lemma which shows that least fixed point recursion is automatically well-defined (and formalises some arguments given in [12]):

**Lemma 4.5** *Let  $\mathcal{M}$  be a compositional pre-model. Let  $\mathcal{C}_1, \mathcal{C}_2$  be CSP contexts, such that for any process  $P$  we have  $\mathcal{M}(\mathcal{C}_1[P]) = \mathcal{M}(\mathcal{C}_2[P])$ . Let the least fixed points of  $\mathcal{C}_1$  and  $\mathcal{C}_2$  (viewed as functions on  $\mathcal{P}(\mathcal{FL})$  under the subset order) be  $P_1$  and  $P_2$  respectively. Then  $\mathcal{M}(P_1) = \mathcal{M}(P_2)$ .*

**Proof.** Using the fact that CSP contexts induce Scott-continuous functions on  $\mathcal{P}(\mathcal{FL})$  (see [6], Section 2.8.2), the Kleene fixed point theorem gives that  $P_i = \bigcup_{n=0}^{\infty} \mathcal{C}_i^n(\perp)$ . Now any  $x \in \mathcal{M}(P_1)$  is in the union taken up to some finite  $N$ , and since finite unions correspond to internal choice, and  $\perp$  to the process **div**, we have that the unions up to  $N$  of  $\mathcal{C}_1$  and  $\mathcal{C}_2$  agree under  $\mathcal{M}$  by compositionality. Hence  $x \in \mathcal{M}(P_2)$ , so  $\mathcal{M}(P_1) \subseteq \mathcal{M}(P_2)$ . Similarly  $\mathcal{M}(P_2) \subseteq \mathcal{M}(P_1)$ .  $\square$

**Definition 4.6** A pre-model  $\mathcal{M}$  is *extensional* if for all alphabets  $\Sigma_1 \subseteq \Sigma_2$  we have that  $\text{obs}_{\Sigma_1}(\mathcal{M}) \subseteq \text{obs}_{\Sigma_2}(\mathcal{M})$ , and  $\mathcal{M}_{\Sigma_2}$  agrees with  $\mathcal{M}_{\Sigma_1}$  on  $\mathcal{FL}(\Sigma_1) \times \text{obs}_{\Sigma_1}(\mathcal{M})$ .

**Definition 4.7** A pre-model is a *model* if it is compositional and extensional.

In this setting, we now describe the five main finite observational models coarser than  $\mathcal{FL}$ : traces, failures, revivals, acceptances and refusal testing.

#### 4.2.1 The traces model

The coarsest model measures only the *traces* of a process; that is, the sequences of events it is able to accept. This corresponds to the language of the process viewed as a nondeterministic finite automaton (NFA).

**Definition 4.8** The *traces* model,  $\mathcal{T}$ , is given by

$$\text{obs}_{\Sigma}(\mathcal{T}) = \Sigma^*, \quad \mathcal{T}_{\Sigma} = \text{trace}_{\Sigma}$$

where *trace* is the equivalence relation which relates the observation  $\langle A_0, a_1, A_1, \dots, a_n, A_n \rangle$  to the string  $a_1 \dots a_n$ .

#### 4.2.2 Failures

The traces model gives us information about what a process is *allowed* to do, but it in some sense tells us nothing about what it is *required* to do. In particular, the process *STOP* trace-refines any other process.

In order to specify liveness properties, we can incorporate some information about the events the process is allowed to refuse, beginning with the *failures* model. Intuitively, this captures traces  $s$ , together with the sets of events the process is allowed to stably refuse after  $s$ .

**Definition 4.9** The *failures* model,  $\mathcal{F}$ , is given by

$$\text{obs}_\Sigma(\mathcal{F}) = \Sigma^* \times (\mathcal{P}(\Sigma) \cup \{\bullet\}), \quad \mathcal{F}_\Sigma = \text{fail}_\Sigma,$$

where  $\text{fail}_\Sigma$  relates the observation  $\langle A_0, \dots, a_n, A_n \rangle$  to all pairs  $(a_1 \dots a_n, X)$ , for all  $X \subseteq \Sigma \setminus A_n$  if  $A_n \neq \bullet$ , and for  $X = \bullet$  otherwise.

#### 4.2.3 Revivals

The next coarsest model, first introduced in [11], is the *revivals* model. Intuitively this captures traces  $s$ , together with sets  $X$  that can be stably refused after  $s$ , and events  $a$  (if any) that can then be accepted.

**Definition 4.10** The *revivals* model,  $\mathcal{R}$ , is given by

$$\text{obs}_\Sigma(\mathcal{R}) = \Sigma^* \times (\mathcal{P}(\Sigma) \cup \{\bullet\}) \times (\Sigma \cup \{\bullet\}), \quad \mathcal{R}_\Sigma = \text{rev}_\Sigma,$$

where  $\text{rev}_\Sigma$  relates the observation  $\langle A_0, a_1, \dots, a_{n-1}, A_{n-1}, a_n, A_n \rangle$  to

- (i) the triples  $(a_1 \dots a_{n-1}, X, a_n)$ , for all  $X \subseteq \Sigma \setminus A_{n-1}$  if  $A_{n-1} \neq \bullet$  and for  $X = \bullet$  otherwise, and
- (ii) the triples  $(a_1 \dots a_n, X, \bullet)$ , for all  $X \subseteq \Sigma \setminus A_n$  if  $A_n \neq \bullet$  and for  $X = \bullet$  otherwise.

A finite linear observation is related to all triples consisting of: its initial trace; a stable refusal that could have been observed, or  $\bullet$  if the original observation did not observe stability; and optionally (part (i) above) a single further event that can be accepted.

#### 4.2.4 Acceptances

All the models considered up to now refer only to sets of refusals, which in particular are closed under subsets. The next model, *acceptances* (also known as ‘ready sets’), refines the previous three and also considers the precise sets of events that can be stably accepted at the ends of traces.

**Definition 4.11** The *acceptances* model,  $\mathcal{A}$ , is given by

$$\text{obs}_\Sigma(\mathcal{A}) = \Sigma^* \times (\mathcal{P}(\Sigma) \cup \{\bullet\}), \quad \mathcal{A}_\Sigma = \text{acc}_\Sigma,$$

where  $\text{acc}_\Sigma$  relates the observation  $\langle A_0, a_1, \dots, a_n, A_n \rangle$  to the pair  $(a_1 \dots a_n, A_n)$ .

#### 4.2.5 Refusal testing

The final model we consider is that of *refusal testing*, first introduced in [9]. This refines  $\mathcal{F}$  and  $\mathcal{R}$  by considering an entire history of events and stable refusal sets. It is incomparable to  $\mathcal{A}$ , because it does not capture precise acceptance sets.

**Definition 4.12** The *refusal testing* model,  $\mathcal{RT}$ , is given by

$$\text{obs}_\Sigma(\mathcal{RT}) = \{\langle X_0, a_1, X_1, \dots, a_n, X_n \rangle : n \in \mathbb{N}, a_i \in \Sigma, X_i \subseteq \Sigma \text{ or } X_i = \bullet\}$$

$$\mathcal{RT}_\Sigma = rt_\Sigma,$$

where  $rt_\Sigma$  relates the observation  $\langle A_0, \dots, a_n, A_n \rangle$  to  $\langle X_0, \dots, a_n, X_n \rangle$ , for all  $X_i \subseteq \Sigma \setminus A_i$  if  $A_i \neq \bullet$ , and for  $X_i = \bullet$  otherwise.

### 4.3 Rational models

We will later on wish to consider only models  $\mathcal{M}$  for which the correspondence between  $\mathcal{FL}$ -observations and  $\mathcal{M}$  observations is decidable by a finite memory computer. We will interpret this notion as saying the relation  $\mathcal{M}_\Sigma$  corresponds to the language accepted by some finite state automaton. In order to do this, we must first decide how to convert elements of  $\mathcal{FL}_\Sigma$  to words in a language. We do this in the obvious way (the reasons for using fresh variables to represent the  $A_i$  will become apparent in Section 5).

**Definition 4.13** The *canonical encoding* of  $\mathcal{FL}_\Sigma$  is over the alphabet  $\Xi := \Sigma \cup \Sigma'' \cup \text{Sym}$ , where  $\Sigma'' := \{a'' : a \in \Sigma\}$  and  $\text{Sym} = \{\langle, \rangle, ', \bullet\}$ .<sup>4</sup> It is given by the representation in Definition 4.1, where sets  $A_i$  are expressed by listing the elements of  $\Sigma''$  corresponding to the members of  $A_i$  in alphabetical order. We denote this encoding by  $\phi_\Sigma : \mathcal{FL}_\Sigma \rightarrow \Xi^*$ .

We now define a model to be *rational* (borrowing a term from automata theory) if its defining relation can be recognised (when suitably encoded) by some nondeterministic finite automaton.

**Definition 4.14** A model  $\mathcal{M}$  is *rational* if for every alphabet  $\Sigma$ , there is a partial order embedding<sup>5</sup>  $\psi_\Sigma : \text{obs}_\Sigma(\mathcal{M}) \rightarrow \Theta^*$  of  $\text{obs}_\Sigma(\mathcal{M})$  in some finite alphabet  $\Theta$ , and a (nondeterministic) finite automaton  $\mathcal{A}$  recognising  $\{(\phi_\Sigma(x), \psi_\Sigma(y)) : (x, y) \in \mathcal{M}_\Sigma\}$ .

What does it mean for an automaton to ‘recognise’ a relation?

**Definition 4.15** For alphabets  $\Sigma$  and  $T$ , a relation  $\mathcal{R} \subseteq \Sigma^* \times T^*$  is *recognised* by an automaton  $\mathcal{A}$  just when:

- (i) The event-set of  $\mathcal{A}$  is  $\text{left}.\Sigma \cup \text{right}.T$ , and
- (ii) For any  $s \in \Sigma^*, t \in T^*$ , we have  $s\mathcal{R}t$  if and only if there is some interleaving of  $\text{left}.s$  and  $\text{right}.t$  accepted by  $\mathcal{A}$ .

Note that  $\mathcal{FL}$  itself (viewing  $\mathcal{FL}_\Sigma$  as the prefix relation) is trivially rational.

**Lemma 4.16** The models  $\mathcal{T}, \mathcal{F}, \mathcal{R}, \mathcal{A}$  and  $\mathcal{RT}$  are rational.

**Proof.** By inspection of Definitions 4.8–4.12. We take  $\Theta = \Sigma \cup \Sigma' \cup \Sigma'' \cup \text{Sym}$ , with  $\Sigma''$  and the expression of acceptance sets as in the canonical encoding of  $\mathcal{FL}$ , and refusal sets expressed in the corresponding way over  $\Sigma' := \{a' : a \in \Sigma\}$ .  $\square$

<sup>4</sup> Note that this somewhat unsatisfactory notation denotes a set of four elements: the angle brackets  $\langle$  and  $\rangle$ , the comma  $,$ , and the symbol  $\bullet$ .

<sup>5</sup> Recall that a *partial order embedding* is a map of partial orders which is an isomorphism onto its image.

Note that not all relations are rational. For instance, the ‘counting relation’ mapping each finite linear observation to its length is clearly not rational. We do not know whether the additional constraint of being a finite observational model necessarily implies rationality; however, no irrational models are known. We therefore venture the following conjecture:

**Conjecture 4.17 (Rationality of finite observational models)** *Let  $\mathcal{M}$  be a finite observational model. Then  $\mathcal{M}$  is rational.*

## 5 Model shifting

We now come to the main substance of this paper: we prove results on ‘model shifting’, showing that there exist contexts allowing us to pass between different semantic models and the basic traces model. The main result is Theorem 5.4, which shows that this is possible for any rational model.

### 5.1 Model shifting for $\mathcal{FL}$

We begin by proving the result for the finest model,  $\mathcal{FL}$ . We show that there exists a context  $\mathcal{C}_{\mathcal{FL}}$  such that for any process  $P$ , the finite linear observations of  $P$  correspond to the traces of  $\mathcal{C}_{\mathcal{FL}}(P)$ .

**Lemma 5.1 (Model shifting for  $\mathcal{FL}$ )** *For every alphabet  $\Sigma$ , there exists a context  $\mathcal{C}_{\mathcal{FL}}$  over alphabet  $T := \Sigma \cup \Sigma' \cup \Sigma'' \cup \{\text{done}\}$ , and a partial order embedding  $\pi : \mathcal{FL}_{\Sigma} \rightarrow T^*$  (with respect to the prefix partial order on each set) such that for any process  $P$  over  $\Sigma$  we have  $\mathcal{T}(\mathcal{C}_{\mathcal{FL}}[P]) = \text{pref}(\pi(\mathcal{FL}(P)))$  (where  $\text{pref}(X)$  is the prefix-closure of the set  $X$ ).*

**Proof.** We will use the unprimed alphabet  $\Sigma$  to denote communicated events from the original trace, and the double-primed alphabet  $\Sigma''$  to denote stable acceptances.  $\Sigma'$  will be used in an intermediate step to denote refusals, and *done* will be used to distinguish  $\emptyset$  (representing an empty acceptance set) from  $\bullet$  (representing a failure to observe anything).

**Step 1:** We first produce a process which is able to communicate events  $x'_i$ , just when the original process can stably refuse the corresponding  $x_i$ . Define the partial order  $\leq_1 = \langle x' <_1 x : x \in \Sigma \rangle$ , which prevents refusal events when the corresponding event can occur.

Let the context  $\mathcal{C}_1$  be given by

$$\mathcal{C}_1[X] = \text{prioritise}(X \parallel \text{RUN}_{\Sigma'}, \leq_1, \Sigma).$$

Note that the third argument prevents primed events from occurring in unstable states.

**Step 2:** We now similarly introduce acceptance events, which can happen in stable states when the corresponding refusal can’t.

Similarly define the partial order  $\leq_2 = \langle x'' <_2 x' : x \in \Sigma \rangle$ , which prevents acceptance events when the corresponding refusal is possible. Let the context  $\mathcal{C}_2$  be defined by

$$\mathcal{C}_2[X] = \text{prioritise}(\mathcal{C}_1[X] \parallel \text{RUN}_{\Sigma''}, \leq_2, \Sigma).$$



**Step 3:** We now ensure that an acceptance set inferred from a trace is a complete set accepted by the process under examination. This is most straightforwardly done by employing a regulator process, which can either accept an unprimed event or accept the alphabetically first refusal or acceptance event, followed by a refusal or acceptance for each event in turn. In the latter case it then communicates a *done* event, and returns to its original state.

The *done* event is necessary in order to distinguish between a terminal  $\emptyset$ , which can have a *done* after the last event, and a terminal  $\bullet$ , which cannot (observe that a  $\emptyset$  cannot occur other than at the end). Finally, we hide the refusal events.

Let  $a$  and  $z$  denote the alphabetically first and last events respectively, and let  $\text{succ } x$  denote the alphabetical successor of  $x$ . Define the processes

$$\begin{aligned} UNSTABLE &= \square_{x \in \Sigma} x \rightarrow UNSTABLE \\ &\quad \square a' \rightarrow STABLE(a) \square a'' \rightarrow STABLE(a) \\ STABLE(x) &= x' \rightarrow STABLE(\text{succ } x) \square x'' \rightarrow STABLE(\text{succ } x) \quad (x \neq z) \\ STABLE(z) &= done \rightarrow UNSTABLE, \end{aligned}$$

and let

$$\mathcal{C}_{\mathcal{FL}}[X] = \left( \mathcal{C}_2[X] \parallel_{\Sigma \cup \Sigma' \cup \Sigma''} UNSTABLE \right) \setminus \Sigma'.$$

**Step 4:** We now complete the proof by defining the function  $\pi$  inductively as follows:

$$\begin{aligned} \pi(s^{\wedge} \langle \bullet \rangle) &= \pi(s) \\ \pi(s^{\wedge} \langle x \rangle) &= \pi(s)^{\wedge} \langle x \rangle \\ \pi(s^{\wedge} \langle A = \{x_1, \dots, x_k\} \rangle) &= \pi(s)^{\wedge} \langle x_1'' \dots x_k'' done \rangle, \end{aligned}$$

where without loss of generality the  $x_i$  are listed in alphabetical order.

It is clear that this is a partial order embedding, and by the construction above satisfies  $\mathcal{T}(\mathcal{C}_{\mathcal{FL}}[P]) = \text{pref}(\pi(\mathcal{FL}(P)))$ .  $\square$

This result allows us to translate questions of  $\mathcal{FL}$ -refinement into questions of trace refinement under  $\mathcal{C}_{\mathcal{FL}}$ , as follows:

**Corollary 5.2** *For  $\mathcal{C}_{\mathcal{FL}}$  as in Lemma 5.1, and for any processes  $P$  and  $Q$ , we have  $P \sqsubseteq_{\text{FL}} Q$  if and only if  $\mathcal{C}_{\mathcal{FL}}[P] \sqsubseteq_{\text{T}} \mathcal{C}_{\mathcal{FL}}[Q]$ .*

**Proof.** Certainly if  $\mathcal{FL}(Q) \subseteq \mathcal{FL}(P)$  then  $\mathcal{T}(\mathcal{C}_{\mathcal{FL}}[Q]) = \text{pref}(\pi(\mathcal{FL}(Q))) \subseteq \text{pref}(\pi(\mathcal{FL}(P))) = \mathcal{T}(\mathcal{C}_{\mathcal{FL}}[P])$  and so  $\mathcal{C}_{\mathcal{FL}}[P] \sqsubseteq_{\text{T}} \mathcal{C}_{\mathcal{FL}}[Q]$ .

Conversely, suppose there exists  $x \in \mathcal{FL}(Q) \setminus \mathcal{FL}(P)$ . Then since  $\mathcal{FL}(P)$  is downwards-closed, we have  $x \not\leq y$  for all  $y \in \mathcal{FL}(P)$ . Since  $\pi$  is a partial order embedding, we have correspondingly  $\pi(x) \not\leq \pi(y)$  for all  $y \in \mathcal{FL}(P)$ . Hence  $\pi(x) \notin \text{pref}(\pi(\mathcal{FL}(P)))$ , so  $\text{pref}(\pi(\mathcal{FL}(Q))) \not\subseteq \text{pref}(\pi(\mathcal{FL}(P)))$ .  $\square$

### 5.2 Model shifting for rational observational models

We now have essentially all we need to prove the main theorem. We record a folk result, that any NFA can be implemented as a CSP process (up to prefix-closure, since trace-sets are prefix-closed but regular languages are not):

**Lemma 5.3 (Implementation for NFA)** *Let  $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$  be a (non-deterministic) finite automaton. Then there exists a CSP process  $P_{\mathcal{A}}$  such that  $\text{pref}(L(\mathcal{A})) = \text{pref}(\mathcal{T}(P_{\mathcal{A}}))$ .*

**Proof.** Trivial construction. See Chapter 7 of [10].  $\square$

**Theorem 5.4 (Model shifting for rational models)** *For every rational model  $\mathcal{M}$ , there exists a context  $\mathcal{C}_{\mathcal{M}}$  such that for any process  $P$  we have  $\mathcal{T}(\mathcal{C}_{\mathcal{M}}[P]) = \text{pref}(\psi(\mathcal{M}(P)))$ .*

**Proof.** Let  $\mathcal{A}$  be the automaton recognising  $(\phi \times \psi)(\mathcal{M})$  (as from Definition 4.14), and let  $P_{\mathcal{A}}$  be the corresponding process from Lemma 5.3.

We first apply Lemma 5.1 to produce a process whose traces correspond to the finite linear observations of the original process, prefixed with left: let  $\mathcal{C}_{\mathcal{FL}}$  be the context from Lemma 5.1, and let the context  $\mathcal{C}_1$  be defined by

$$\mathcal{C}_1[X] = \mathcal{C}_{\mathcal{FL}}[X][\text{left}.x/x].$$

We now compose in parallel with  $P_{\mathcal{A}}$ , to produce a process whose traces correspond to the  $\mathcal{M}$ -observations of the original process. Let  $\mathcal{C}_2$  be defined by

$$\mathcal{C}_2[X] = \left( \left( \mathcal{C}_1[X] \parallel_{\{\text{left}\}} P_{\mathcal{A}} \right) \setminus \{\text{left}\} \right) [x/\text{right}.x].$$

Then the traces of  $\mathcal{C}_2[X]$  are precisely the prefixes of the images under  $\psi$  of the observations corresponding to  $X$ , as required.  $\square$

By the same argument as for Corollary 5.2, we have

**Corollary 5.5** *For any rational model  $\mathcal{M}$ , let  $\mathcal{C}_{\mathcal{M}}$  be as in Theorem 5.4. Then for any processes  $P$  and  $Q$ , we have  $P \sqsubseteq_{\mathcal{M}} Q$  if and only if  $\mathcal{C}_{\mathcal{M}}[P] \sqsubseteq_{\text{T}} \mathcal{C}_{\mathcal{M}}[Q]$ .*

## 6 Implementation

We demonstrate the technique by implementing contexts with the property of Corollary 5.5; source code may be found at [1].

For the sake of efficiency we work directly rather than using the general construction of Theorem 5.4. The context **C1** introduces refusal events and a **stab** event, which can occur only when the corresponding normal events can be refused. This implements the refusal testing model, and the context **CF** which allows only normal events optionally followed by some refusals (and **stab**) implements the failures model.

This is however suboptimal over large alphabets, in the typical situation where most events are refused most of the time. FDR3's inbuilt failures refinement checking is able to compare *acceptance* sets (checking that the acceptances of the spec-

ification are a subset of those of the implementation), which are typically smaller than the refusal sets.

The context  $\mathbf{C}'$  introduces acceptance events which can occur only in stable states where the corresponding refusal cannot, and then blocks all refusals. The problem then is: how to check that the acceptances of the *specification* are a subset of those of the *implementation*, despite the fact that trace refinement checks for inclusion the other way?

The answer is to use priority to prevent the **stab** event from happening while acceptances are still available, so that  $\mathbf{CFImpl}'$  is able to communicate only its *precise* acceptance sets. We then form  $\mathbf{CFSpec}'$  by parallel composition with  $\mathbf{RUN}$  for all the acceptance events, so that  $\mathbf{CFSpec}'$  can communicate any *supersets* of its acceptance set.

Similar constructions with slightly different restrictions on the permissible sequences of events produce efficient processes for the revivals and refusal testing models. For the acceptances model, we just want to check for inclusion of the implementation's acceptance sets in those of the specification, so the context  $\mathbf{CFImpl}'$  works for both the specification and the implementation; finite linear observations works similarly with failures replaced by refusal testing.

### 6.1 Testing

We test this implementation by constructing processes which are first distinguished by the failures, revivals, refusal testing and acceptance models respectively (the latter two being also distinguished by the finite linear observations model). The processes, and the models which do and do not distinguish them, are shown in Table 1 (recall the precision hierarchy of models:  $\mathcal{T} \leq \mathcal{F} \leq \mathcal{R} \leq \{\mathcal{A}, \mathcal{RT}\} \leq \mathcal{FL}$ ). The correct results are obtained when these checks are run in FDR3 with the implementation described above.

Specification	Implementation	Passes	Fails
$a \rightarrow \mathbf{div}$	$a \rightarrow \mathbf{STOP}$	$\mathcal{T}$	$\mathcal{F}$
$((a \rightarrow \mathbf{div}) \sqcap \mathbf{div}) \sqcap \mathbf{STOP}$	$a \rightarrow \mathbf{div}$	$\mathcal{F}$	$\mathcal{R}$
$(a \rightarrow \mathbf{div}) \sqcap (\mathbf{div} \triangle (a \rightarrow \mathbf{STOP}))$	$a \rightarrow \mathbf{STOP}$	$\mathcal{R}, \mathcal{A}$	$\mathcal{RT}, \mathcal{FL}$
$(a \rightarrow \mathbf{STOP}) \sqcap (b \rightarrow \mathbf{STOP})$	$(a \rightarrow \mathbf{STOP}) \sqcap (b \rightarrow \mathbf{STOP})$	$\mathcal{R}, \mathcal{RT}$	$\mathcal{A}, \mathcal{FL}$

Table 1

Tests distinguishing levels of the model precision heirarchy.  $\triangle$  is the *interrupt* operator; see [12] for details.

### 6.2 Performance

We assess the performance of our simulation by running those examples from Table 1 of [5] which involve refinement checks (as opposed to deadlock- or divergence-freedom assertions), and comparing the timings for our construction against the time taken by FDR3's inbuilt failures refinement check (since  $\mathcal{F}$  is the only model for which we have a point of comparison between a direct implementation and the methods developed in this paper). Results are shown in Table 2, for both the original and revised contexts described above; the performance of the  $\mathcal{FL}$  check is also shown. As may be seen, performance is somewhat worse but not catastrophically

so. Note however that these processes involve rather small alphabets; performance is expected to be worse for larger alphabets.

Input File	Inbuilt $\mathcal{F}$			CF			CF'			FL		
	$ S $	$ \Delta $	$T(s)$	$ S $	$ \Delta $	$T(s)$	$ S $	$ \Delta $	$T(s)$	$ S $	$ \Delta $	$T(s)$
<b>inv</b>	21M	220M	23	21M	220M	78	21M	220M	125	21M	220M	145
<b>nspk</b>	6.9M	121M	22	6.3M	114M	73	4.1M	72M	55	5.4M	97M	92
<b>swp</b>	24M	57M	16	30M	123M	61	43M	76M	107	42M	93M	131

Table 2

Experimental results comparing the performance of our construction with FDR3's inbuilt failures refinement check.  $|S|$  is the number of states,  $|\Delta|$  is the number of transitions,  $T$  is the time (in seconds), and  $M$  indicates millions.

### 6.3 Example: Conflict detection

We illustrate the usefulness of richer semantic models than just traces and failures by giving a sample application of the revivals model. Suppose that we have a process  $P$  consisting of the parallel composition of two sub-processes  $Q$  and  $R$ . The failures model is able to detect when  $P$  can refuse all the events of their shared alphabet, or deadlock in the case when they are synchronised on the whole alphabet. However, it is unable to distinguish between the two possible causes of this: it may be that one of the composands is able to refuse the entire shared alphabet, or it may be that each accepts some events from the shared alphabet, but the acceptances of  $Q$  and  $R$  are disjoint. We refer to the latter situation as a ‘conflict’. The absence of conflict (and similar situations) is at the core of a number of useful ways of proving deadlock-freedom for networks of processes running in parallel [14].

The revivals model can be used to detect conflicts. For a process  $P = Q \parallel_X R$ , we introduce a fresh event  $a$  to represent a generic event from the shared alphabet, and form the process  $P' = Q' \parallel_{X'} R'$ , where  $Q' = Q \parallel_{\{(x,x), (x,a) : x \in X\}}$ ,  $X' = X \cup \{a\}$ , and similarly for  $R'$  and  $Y'$ . Conflicts of  $P$  now correspond to revivals  $(s, X \cap Y, a)$ , where  $s$  is a trace not containing  $a$ .

## 7 Conclusions

The result of Theorem 5.4 shows that the expressibility of all finite observational (rational) models can in some sense be simulated by the traces model using the priority operator. This provides a practical method of testing refinement over models that FDR does not directly support. While any such model could be implemented directly in the program itself, we have shown this is not necessary. This also serves to further demonstrate the power and usefulness of the priority operator (see also the previous work of the second-named author on the expressiveness of CSP with priority [13] and on ‘slow abstraction’ [15]).

Note that this type of construction can be used more generally. Firstly, it seems likely that the construction can be extended to non-finite models; for instance to reduce failures-divergences tests to traces-divergences, or infinite-traces-failures-divergences to infinite-traces-divergences.

Secondly, the construction does not use the requirement that a model be compositional. This means that it will work for any rational set of observable be-

haviours, such as the singleton failures semantics presented in [3]. The techniques described here can also be used to support the Timed Failures model of Timed CSP in FDR3 [2].

The limitation to rational models is from a theoretical point of view rather unsatisfactory, although it may be of little practical significance since all known models (and probably all models one would be likely to come up with) are clearly rational. However, Conjecture 4.17 remains of interest since a resolution in either direction would undoubtedly yield insight into the structure of the ‘clouds’ of models lying above  $\mathcal{R}$  set out in [11].

### Acknowledgements

The authors are grateful to Tom Gibson-Robinson for helpful discussions and practical assistance with FDR3. This work has been partially sponsored by DARPA under agreement number FA8750-12-2-0247.

## References

- [1] [www.cs.ox.ac.uk/people/david.mestel/model-shifting.csp](http://www.cs.ox.ac.uk/people/david.mestel/model-shifting.csp).
- [2] Philip Armstrong, Gavin Lowe, Joël Ouaknine, and Bill Roscoe. Model checking timed CSP. In Andrei Voronkov and Margarita Korovina, editors, *HOWARD-60. A Festschrift on the Occasion of Howard Barringer’s 60th Birthday*, pages 13–33. EasyChair, 2014.
- [3] Christie Bolton and Jim Davies. A singleton failures semantics for communicating sequential processes. *Formal Aspects of Computing*, 18(2):181–210, 2006.
- [4] Thomas Gibson-Robinson, Philip Armstrong, Alexandre Boulgakov, and A.W. Roscoe. FDR3—a modern refinement checker for CSP. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 187–201. Springer, 2014.
- [5] Thomas Gibson-Robinson, Henri Hansen, A.W. Roscoe, and Xu Wang. Practical partial order reduction for CSP. In *NASA Formal Methods*, pages 188–203. Springer, 2015.
- [6] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.
- [7] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.
- [8] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Information and Computation*, 100(1):1–40, 1992.
- [9] Iain Phillips. Refusal testing. *Theoretical Computer Science*, 50(3):241–284, 1987.
- [10] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.
- [11] A.W. Roscoe. Revivals, stuckness and the hierarchy of CSP models. *The Journal of Logic and Algebraic Programming*, 78(3):163–190, 2009.
- [12] A.W. Roscoe. *Understanding Concurrent Systems*. Texts in Computer Science. Springer, 2010.
- [13] A.W. Roscoe. The expressiveness of CSP with priority. In *Proceedings of MFPS 2015*, 2015.
- [14] A.W. Roscoe and Naiem Dathi. The pursuit of deadlock freedom. *Information and Computation*, 75(3):289 – 327, 1987.
- [15] A.W. Roscoe and Philippa J. Hopcroft. Theories of programming and formal methods. chapter Slow Abstraction via Priority, pages 326–345. Springer-Verlag, Berlin, Heidelberg, 2013.

# On the Representation of Semigroups and Other Congruences in the Lambda Calculus

Rick Statman  
Carnegie Mellon University  
Department of Mathematical Sciences  
Pittsburgh, PA 15213  
statman@cs.cmu.edu

May 20, 2016

## Abstract

Lambda Calculus is the starting point of all functional programming. Since Church noticed the undecidability of the word problem for semigroups ([2]), it has been understood that certain algebraic structures are embedded in lambda calculus. These include the  $B, I$  monoid ([3], [4]), the free Cartesian monoid which includes the positive part of the Freyd, Heller, Thompson group ([6], [8]), and the near semi-rings and b.a.d. algebras of [9].

We show that every semigroup with an RE word problem can be pointwise represented in the lambda calculus. In addition, we show that the free monoid generated by an arbitrary RE subset of combinators can be represented as the monoid of all terms which fix a finite set of points.

Combinators being both functions and arguments can act on one another by application and composition. More generally, if  $\$'$  and  $\$''$  are sets of combinators closed under beta conversion, the  $A$  action of  $\$'$  on  $\$''$  is the set  $\{AMN \mid M : \$' \text{ and } N : \$''\}$  closed under beta conversion. First we recall the definitions of some familiar combinators

$B := \lambda abc.a(bc)$   
 $B' := \lambda abc.b(ac)$   
 $C^* := \lambda ab.ba$   
 $K := \lambda ab.a$   
 $I := \lambda a.a$   
 $S := \lambda abc.ac(bc)$   
 $O := (\lambda x.xx)(\lambda x.xx)$   
 $0 := \lambda yz.z$   
 $s := \lambda xyz.y(xyz)$   
 $Y := (\lambda xz.z(xxz))(\lambda xyz.z(xxz))$

**Examples:**

- (1)  $A := K$  : this is the trivial action.
- (2)  $A := I$  : this is the applicative action.
- (3)  $A := B$  : this is the semigroup action.
- (4)  $A := S$  : the pointwise applicative action.

Of course, this definition extends to multiple arguments by Curryng. We write

$$M = N \text{ mod beta}$$

if  $M$  beta converts to  $N$ .

It is trivial that general  $A$  can be reduced to  $I$ , and that multiple arguments can be reduced to a single argument by pairing. In addition, applicative action can be reduced to the semigroup action since  $K(xy) = Bx(Ky)$  mod beta. However, there is another reduction which is lambda  $I$ .

Let

$$D := Y(\lambda fxyz. fx(zy))$$

where  $Y$  is Turing's fixed point combinator as above.

**Lemma 1.** *For any  $U, V$  if  $B(C^*U)D = B(C^*V)D$  mod beta then*

$$U = V \text{ mod beta.}$$

*Proof.* **Straight forward.**

□

Now given  $\$'$  and  $\$''$ , since

$$\begin{aligned} C^*(AM) &= B(B(C^*M)(C^*A))B \text{ mod beta} \\ C^*(AMN) &= B(B(B(C^*M)(C^*A))B)(B(C^*N)D) \text{ mod beta, and} \\ &= B(C^*M)(B(C^*A)(BB(B(C^*N)D))) \text{ mod beta} \end{aligned}$$

the  $A$  action of  $\$'$  on  $\$''$  is equivalent to the semigroup action of  $\{C^*M \mid M : \$'\}$  on  $\{(B(C^*A)(BB(B(C^*N)D))) \mid N : \$''\}$ . We next consider an example of the action of  $I$  in representing semi-groups.

**Definition:** Let  $\$'$  be an RE set of combinators closed under beta conversion. An equivalence relation  $\sim$  on  $\$'$  is said to be pointwise representable on  $\$''$  if for every  $M, N : \$'$  we have

$$MP : \$'' \text{ for all } P : \$''$$

$$\begin{aligned} M \sim N &\text{ iff for all } P : \$'' \\ &MP = NP \text{ mod beta} \end{aligned}$$

**Example (Kleene):**

$$\begin{aligned} \$' &= \text{any RE set of definitions of total recursive functions} \\ \$'' &= \text{the Church numerals and } \sim = \text{extensional equality} \end{aligned}$$

**Non-example (Plotkin):**

$$\begin{aligned} \$' &= \text{all combinators} \\ \$'' &= \text{all combinators and } \sim = \text{beta conversion.} \end{aligned}$$

Let  $\$$  be a semigroup on a countable number of generators. We assume that the generators are denoted by the positive integers. Elements of  $\$$  are then denoted by words

$$w = w(1) \dots w(l).$$

of variable length  $l$ , on the generators of  $\$$ . We write  $u = v \text{ mod } \$$  if the words  $u$  and  $v$  are equal in  $\$$ . We represent the word  $w$  by the lambda term

$$'w' := B'Ow(1)(B'Ow(2)(\dots B'Ow(n-1)Ow(n)\dots))$$

where integers are replaced by their Church numerals.

We define combinators  $P, Q, R$  as follows.



By Theorem 3 of [7] there exists a closed term  $P$  such that  $PM = PN \text{ mod beta}$  if and only if either  $M = N \text{ mod beta}$  or both  $M = 'u'U \text{ mod beta}$ ,  $N = 'v'V \text{ mod beta}$ , and  $u = v \text{ mod \$}$ , where either  $U$  or  $V$ , or both, may not exist, but each must be of positive order if it exists. Now we set

$$\begin{aligned} p &:= \text{predecessor for Church numerals, and} \\ A &:= Y(\lambda xy. y0(B(py)1) \text{ (An} = n^{\text{th}} \text{ eta expansion of } I \text{ mod beta)} \\ Q &:= \lambda xy. Y(Ax(fsx)(Py)) \\ R &:= Q0. \end{aligned}$$

For each word  $w$  we define a second representation by the lambda term

$$''w'' := B(C^* 'w')B.$$

Then  $''w'' = \lambda ab.a('w'b) \text{ mod beta}$  and for words,  $w, u$   
 $B''w''''u'' = ''wu'' \text{ mod beta}$

and for any words  $w_1, \dots, w_n, u_1, \dots, u_m$

$$\begin{aligned} ''w''(R''w_1'' \dots ''w_n''(R''u_1'' \dots ''u_m'')) &= \\ Q(n+1)(P(''w_1'')) \dots (P(''w_n''))(P''w''((R''u_1'' \dots ''u_m''))) &= \\ Q(n+1)(P(''w_1'')) \dots (P(''w_n''))(P''w'') \text{ mod beta.} \end{aligned}$$

Now it is not difficult to prove that if

$$\begin{aligned} Q(n+1)(P(''w_1'')) \dots (P(''w_n''))(P''w'') &= \\ Q(n+1)(P(''w_1'')) \dots (P(''w_n''))(P''u'') \text{ mod beta} \end{aligned}$$

then  $w = u \text{ mod \$}$ .

Now take for  $\$'$  the set of all  $''w''$  and for  $\$''$  the set of all  $R(''w_1'') \dots (''w_n'')$ . Thus we have proved the

**Proposition 1.** *Every RE semigroup is pointwise representable.*

For a general RE congruence  $\sim$ , we illustrate with the case of one binary function symbol  $f$ . We assume that we have Gödel numbering  $'t', 'r'$  of terms  $t, r$  with a recursive function  $t, r \mapsto ftr$  represented by a lambda term  $F$ ; that is  $F't'r' = 'ftr' \text{ mod beta}$ . By Theorem 3 of [7] there exists a closed term  $P$  such that  $PM = PN \text{ mod beta}$  if and only if either  $M = N \text{ mod beta}$  or both  $M = 'u' \text{ mod beta}$ ,  $N = 'v' \text{ mod beta}$  and  $u = v \text{ mod \$}$ . Now define an app

$$A := \lambda abcde. < a, e >$$

and define

$$"t'' := \langle A, 't', P't' \rangle$$

$$"f'' := \lambda xy. \langle A, F(xK)(yK), P(F(xK)(yK)) \rangle.$$

The set  $\$''$  can be taken to be the set of all terms  $\langle A, P't' \rangle$ . Thus,

**Proposition 2.** *Every RE congruence is pointwise representable.*

These representation results implicitly use the “regularity” of the representation. If the representing functions are essentially irregular and beta conversion on that set is decidable, such as the set of Church numerals, then co-RE congruences can be represented. Using Kleene brackets  $\{e\}$  for the recursive function with Gödel number  $e$ , we have

**Lemma 2.** *Let  $\sim$  be a co-RE equivalence relation on the set of natural numbers. Then there exists a recursive function  $f$  such that for any  $e$ ,  $\{f(e)\}$  is total recursive and  $i \sim j$  iff  $\{f(i)\} = \{f(j)\}$ .*

*Proof.* We proceed recursively assuming that  $f(i)$  is defined for  $i = 0, \dots, n$ . To define  $f(n+1)$  we compute successive values  $\{f(n+1)\}(j)$  for  $j = 0, \dots, k$ . Assume that these have been computed up to  $k$ . To compute the value for  $k+1$  let  $@$  be the subset of  $\{0, 1, \dots, n\}$  such that  $i : @$  iff there is not  $j < k+1$  with  $\{f(i)\}(j) \neq \{f(n+1)\}(j)$ . Now compute  $\{f(i)\}(k+1)$  for each  $i : @$ . The values partition  $@$ ;  $i$  and  $j$  belong to the same block iff  $\{f(i)\}(k+1) = \{f(j)\}(k+1)$ . Now the set of all  $i$  such that  $i$  is inequivalent to  $n+1$  is uniformly RE in  $n+1$ . For any two distinct blocks in the partition of  $@$ , eventually every member of at least one of the blocks will appear in the enumeration. When there is only one block left in the partition we can set  $\{f(n+1)\}(k+1) = \{f(i)\}(k+1)$  any  $i$  in that block provided after  $k+1$  steps in the enumeration of the inequivalents to  $n+1$  at least one member of that block has not been found. Otherwise, we set  $\{f(n+1)\}(k+1) = 1 + \max[\{f(i)\}(k+1) | i : @]$ . End of proof.  $\square$

The construction for Proposition 2 can now be modified to give

**Proposition 3.** *Every co-RE congruence is pointwise representable.*

Next we consider the case of a general RE set  $\$$  closed under beta conversion. The members of  $\$$  generate a free monoid under the map

$$M \mapsto C^*M.$$

Here we intend to include the Church numeral  $1 = I \text{ mod eta}$  as well as  $I$ . If  $\$''$  is a set of terms closed under beta conversion we say that  $\$'$  is fixed-pointwise representable on  $\$''$  if the set  $\{L \mid LX = X \text{ mod beta for all } X : \$''\}$  = the free monoid generated by

$$\{L \mid L = C^*M \text{ mod beta for some } M : \$'\}$$

Note here that we have specifically allowed  $\$''$  to contain open terms. We recall some of the definitions of [5] with a few small changes.  $T$  is the fixed point combinator of Bohm ([1] 6.5.4) with a free variable  $b$ ;

$$\begin{aligned} E &:= \text{the enumerator of } \{C^*N \mid M : \$\} \\ T &:= (\lambda xyz. z(xxyz))(\lambda xyz. z(xxyz))b \\ A' &:= \lambda fg. \lambda xyz. fx(ax)(f(Sx)y(g(Sx))z) \\ A'' &:= \lambda fg. \lambda x. f(Sx)(a(E(Sx))(g(Sx))(gx)) \\ G &:= T(\lambda u. A''(T(\lambda v. A'vu))u) \\ F &:= T(\lambda u. A'uG) \\ H &:= \lambda xa. F0(ax)(G0) \\ J &:= Y(\lambda f. \lambda xy. f(x(Hy)))(Y(\lambda g. g(H(E0)))) \\ L &:= Y(\lambda fxy. f(x(Jy)) \\ P &:= Y(\lambda f. fJ) \\ Q &:= LP \\ L' &:= Y(\lambda f. \lambda xy. \langle f, x \rangle) \\ L'' &:= Y(\lambda f. \lambda xyz. \langle f, x, z \rangle). \end{aligned}$$

as in [3] have

**Lemma 3.**  $JM = J \text{ mod beta}$  iff there exists an  $m$  such that  $Em = C^*M \text{ mod beta}$ .

Now consider the following “points fixed” equations

- (1)  $x\langle L', 0 \rangle = \langle L', 0 \rangle$
- (2)  $x\langle L'', 0, 1 \rangle = \langle L'', 0, 1 \rangle$
- (3)  $xQ = Q$ .

Now if

$$\begin{aligned}
M &= \lambda a. a(C^*M_1) \dots (C^*M_m) \text{ for } M_i : \$ \text{ then} \\
M\langle L', 0 \rangle &= L'M_1 0(C^*M_2) \dots (C^*M_m) \\
&= \langle L', 0 \rangle (C^*M_2) \dots (C^*M_m) = \dots \\
&= \langle L', 0 \rangle \text{ mod beta, and similarly}
\end{aligned}$$

$$M\langle L'', 0, 1 \rangle = \langle L'', 0, 1 \rangle \text{ mod beta}$$

In addition,

$$\begin{aligned}
MQ &= LP(C^*M_1) \dots (C^*M_m) = L(P(J(C^*M_1)))(C^*M_2) \dots (C^*M_m) \\
&= L(PJ)(C^*M_2) \dots (C^*M_m) = LP(C^*M_2) \dots (C^*M_m) = \dots \\
&= Q \text{ mod data}
\end{aligned}$$

Thus all the members of the free monoid generated by the  $C^*M$  with  $M : \$$  satisfy (1), (2), and (3).

**Proposition 4.** *Suppose that  $N$  satisfies the equations (1), (2), and (3) mod beta, then  $N$  lies in the free monoid generated by the  $C^*M$  for  $M$  in  $\$$ .*

*Proof.* Suppose that such an  $N$  is given. Since  $N$  satisfies (1) mod beta  $N$  has a head normal form. W.l.o.g. we may assume  $N$  is in head normal form. Since  $N$  satisfies equation (2) mod beta, and  $\langle L', 0 \rangle, \langle L'', 0, 1 \rangle$  have head variables with a different number of arguments, the head variable of  $N$  is the first one bound in its lambda prefix. Since  $\langle L', 0 \rangle$  has order 1, the lambda prefix of  $N$  has length 1 or 2. First suppose that  $N$  has order 2:  $N = \lambda xy. xX_1 \dots X_m$ . Then setting  $Z_i := [Q/x]X_i$

$$NQ = \lambda y. QZ_1 \dots Z_m = \lambda y. L(P(JZ_1)) \dots (JZ_m) \text{ mod beta.}$$

By an argument similar to the argument of [7] Theorem 3, this can only be the case if  $Z_m = y \text{ mod beta}$  and for  $i < m$  we have  $JZ_i = J \text{ mod beta}$ . Since  $Q$  contains an unprojectible free variable in  $F$  and  $G$  it must be the case that each  $Z_i$  beta converts to a term without  $x$ , and for  $i < m$  without  $y$ . In other words,  $x$  is head original and thus we assume that

$$N = \lambda xy. xN_1 \dots N_{m-1}y.$$

Hence, by Lemma 3 there exist  $M_1, \dots, M_{m-1} : \$$  such that  $N_i = C^*M_i \text{ mod beta}$  for  $i = 1, \dots, m-1$ , and we have  $N = B1(B(C^*M_1)(\dots (B(C^*M_{m-2})C^*M_{m-1} : \$)) \dots) \text{ mod beta}$ . The case for  $N$  of order 1 is similar with  $I$  replacing 1. End of proof.  $\square$

**Remark:** If the members of  $\$$  all have normal forms then the members of  $\$''$  can be taken to be closed terms.

## References

- [1] Barendregt, H., The Lambda Calculus, North Holland (1982)
- [2] Church, A., A Note on the Entscheidungs Problem, *J. of Symbolic Logic*, **1** (1936).
- [3] Curry, H. B., Feys, R., “Combinatory Logic Vol.I”, North Holland, (1958).
- [4] Statman, R., Combinators and the theory of partitions, CMU Research Report No. 88-31, (1988).
- [5] Statman, R., Some Examples of Non-Existent combinators, *Theoretical Computer Science*, **121**, (1993).
- [6] Statman, R., On Cartesian Monoids, CSL '97 LNCS 1258.
- [7] Statman, R., Morphisms and Partitions of  $V$ -sets, *CSL*, (1998).
- [8] Statman, R., Cartesian Monoids, MFPS 2010 ENTCS Vol 65, Sept. 6, 2010.
- [9] Statman, R., Near semirings and lambda calculus, *TLCA* (2014).

# The algebra of partial equivalence relations

Fabio Zanasi

*Radboud University Nijmegen, The Netherlands*

---

## Abstract

Recent work by the author with Bonchi and Sobociński shows how PROPs of linear relations (subspaces) can be presented by generators and equations via a “cube construction”, based on letting very simple structures interact according to PROP operations of sum, fibered sum and composition via a distributive law. This paper shows how the same construction can be used in a cartesian setting to obtain presentations by generators and equations for the PROP of equivalence relations and of partial equivalence relations.

*Keywords:* PROP, distributive law, string diagram, partial equivalence relation, Frobenius algebra

---

## 1 Introduction

PROPs (**p**roduct and **p**ermutation categories [21]) are symmetric monoidal categories with objects the natural numbers. In the last two decades, they have become increasingly popular as an environment where to study diverse computational models in a compositional, resource sensitive fashion. To make a few examples, they have recently featured in algebraic approaches to Petri nets [7,26], bigraphs [8], quantum processes [11] and signal flow graphs [2,4,1].

PROPs can be used to specify both the *syntax* and the *semantics* of systems. A “syntactic” PROP  $\mathcal{T}$  is generated starting from a *symmetric monoidal theory*  $(\Sigma, E)$ , which intuitively is an algebraic specification for operations with multiple inputs and outputs; arrows of  $\mathcal{T}$  are freely constructed by composition of operations in the signature  $\Sigma$ , and then quotiented by the equations in  $E$ . On the other hand, a “semantic” PROP  $\mathbf{S}$  is specified with a direct definition of its arrows, typically in terms of some mathematical object of interest. A full completeness result is a precise correspondence between these two perspectives, in the form of an isomorphism

$$\mathcal{T} \xrightarrow{\cong} \mathbf{S}. \quad (1)$$

In this situation, we say that  $(\Sigma, E)$  *presents*  $\mathbf{S}$ . Examples of (1) are ubiquitous and play a foundational role in most of the aforementioned research threads. For instance, the theory of commutative monoids presents the PROP of functions; the theory of Hopf algebras presents the PROP of integer matrices; the theory of Frobenius algebras presents the PROP of 2-Dimensional cobordisms.

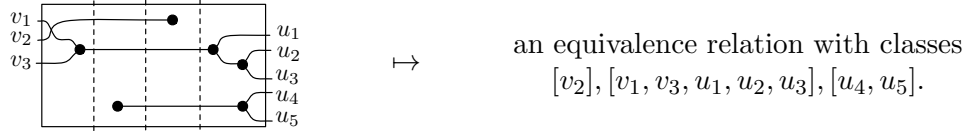
*This paper is electronically published in  
Electronic Notes in Theoretical Computer Science  
URL: [www.elsevier.nl/locate/entcs](http://www.elsevier.nl/locate/entcs)*

In recent years, increasingly more elaborated examples have been tackled using *modular* reasoning principles. An illustrative case is the theory of interacting Hopf algebras  $\mathbf{IH}$ , which characterises the PROP  $\mathbf{LRel}_k$  of  $k$ -linear relations [5]. This result inspired recent investigation in the foundations of the ZX-calculus [3,14] and in categorical control theory [2,4,1]. What is most interesting for our purposes is that the isomorphism  $\mathbf{IH} \cong \mathbf{LRel}_k$  can be obtained as a universal arrow through a “cube” construction, based on seeing the two PROPs as the result of the interaction of simpler theories by means of operations of sum, fibered sum and composition. This modular account is a valuable source of information about the structural properties of the theories of interest: for instance, it shows that  $\mathbf{LRel}_k$  is the result of combining PROPs of spans and of cospans of linear maps, and the equations of  $\mathbf{IH}$  essentially describe this interaction.

The central idea of this work is to show how the same cube construction can be used to characterise other PROPs of relations: whereas [5] focuses on the *linear* case, we shall study the *cartesian* case, both total and partial. In the total case, we construct a modular characterisation for the PROP  $\mathbf{ER}$  of *equivalence relations* starting from PROPs of spans and cospans of (injective) functions, see (5) below. This will show an isomorphism between  $\mathbf{ER}$  and the PROP  $\mathbf{IFr}$  freely generated by a quotient of the theory of special Frobenius algebras [9], which plays a foundational role in many recent works [23,2,1,11].

$$\mathbf{IFr} \xrightarrow{\cong} \mathbf{ER} \quad (2)$$

To give an idea of how the isomorphism (2) works, an arrow of  $\mathbf{IFr}$ , for which we shall use the 2-dimensional representation as a string diagram, as on the left below, shall represent an equivalence relation on the sets of variables associated with its left and right ports, as on the right below. Two variables are in the same equivalence class if they are linked in the graphical representation.



The dotted lines hint at the fact that, as a result of our modular perspective, any diagram of  $\mathbf{IFr}$  will enjoy a factorisation in terms of simpler theories, whose interaction is what the axioms of  $\mathbf{IFr}$  describe.

Building on this result, we will shift to the *partial* case. First, we use PROP composition to construct a presentation  $\mathbf{PMn}$  (**p**artial **c**ommutative **m**onoids) for the PROP  $\mathbf{PF}$  of partial functions. Then, we will show that the PROP  $\mathbf{PER}$  of *partial equivalence relations* (PERs)<sup>1</sup> arises as the result of merging PROPs of cospans of partial functions and of spans of injective functions, see (10) below. As for the case of  $\mathbf{ER}$ , an isomorphism arises from this modular account: it will relate  $\mathbf{PER}$  and the syntactic PROP  $\mathbf{IPFr}$ , yet another variation of the theory of special Frobenius algebras.

$$\mathbf{IPFr} \xrightarrow{\cong} \mathbf{PER}$$

<sup>1</sup> Recall that a relation on a set  $X$  is a PER if it is symmetric and transitive — equivalently, if it is an equivalence relation on a subset  $Y \subseteq X$ .

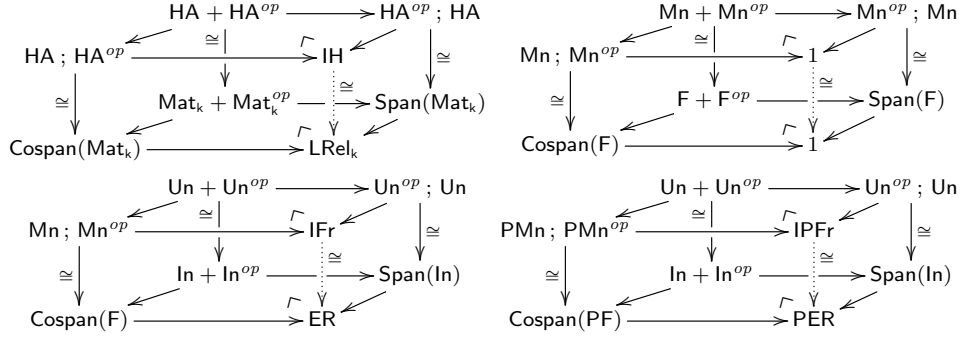


Figure 1. An overview of the various cube constructions considered in this paper. From the top-left corner: the linear case, yielding a characterisation for the PROP  $\mathbf{LRel}_k$  of  $k$ -linear relations (see [5]); the degenerate cartesian case, collapsing to the terminal PROP (Remark 4.11); the (non-degenerate) cartesian case, yielding a characterisation for the PROP  $\mathbf{ER}$  of equivalence relations (Theorem 4.2); the partial cartesian case, yielding a characterisation for the PROP  $\mathbf{PER}$  of partial equivalence relations (Theorem 5.4). In the main text we shall write PROPs of spans and cospans in factorised form to emphasise their provenance from distributive laws, e.g.  $\mathbf{Span}(F)$  as  $F^{op}; F$  and  $\mathbf{Cospan}(F)$  as  $F; F^{op}$ .

In a nutshell, the diagrammatic rendition of partial equivalence relations given by  $\mathbf{IPFr}$  enhances the total case by integrating connectors  $\boxed{\bullet}$ ,  $\boxed{\circ}$  for partiality.

**Related work.** The use of partial equivalence relations in program semantics dates back to the seminal work of Scott [24]. They have been used extensively in the semantics of higher order  $\lambda$ -calculi (e.g., [17,28]) and, more recently, of quantum computations (e.g., [18,15]). Note that in most of these applications PERs are the objects of the category of interest, whereas in the PROP  $\mathbf{PER}$  they are the arrows, with relational composition, and only defined on *finite* domains. In fact, our emphasis is on the modular techniques to characterise  $\mathbf{PER}$  (and their applicability to similar families of structures) rather than on the use of PERs in semantics.

Algebraic presentations for categories of equivalence relations have been studied in the last two decades by a few authors. A characterisation for  $\mathbf{ER}$  in terms of Frobenius structures is given in [13], with a proof based on finding a normal form for string diagrams. The same result appears in a recent manuscript [12], which is based, like our work, on treating equivalence relations as jointly-epi cospans. This idea, as well as its algebraic implications, is studied in the earlier paper [6] as part of a taxonomy of span/cospan categories over  $\mathbf{Set}$ .

The present work is part of the author’s PhD thesis [29], defended in October 2015. Differently from the aforementioned papers, our approach focuses on a modular reconstruction of  $\mathbf{ER}$ : its presentation is built from the interaction of very simple algebraic theories, by the use of PROP operations. In particular, Lack’s technique for composing PROPs [19] is pivotal. Also, we extend our methodology to the analysis of partial functions and partial equivalence relations, in a way that to the best of our knowledge did not appear before in the literature.

It is also worth mentioning that there is a pleasant symmetry between the analysis of equivalence relations and (plain) relations. Whereas the former are jointly-epi cospans and are modeled by separable Frobenius algebras with an additional axiom from the theory of bialgebras, the latter are jointly-mono spans and are modeled by bialgebras with the addition of an axiom from the theory of separable Frobenius



algebras [20]. Interestingly, the combination of the two theories in their entirety collapses to the terminal PROP, see Remark 4.11 below.<sup>2</sup>

**Synopsis.** In §2 we recall the basics of the theory of PROPs. §3 introduces the PROP operations of sum, fibered sum and (iterated) composition, with the example of partial functions (Ex. 3.3). §4 constructs the cube (5) necessary for the characterisation of equivalence relations (Th. 4.2). §5 completes the picture with the characterisation (10) of partial equivalence relations (Th. 5.4).

**Prerequisites and notation.** We assume familiarity with basic category theory (see e.g. [22]) and the definition of symmetric strict monoidal category [22, 25] (often abbreviated as SMC). We write  $f; g: a \rightarrow c$  for composition of  $f: a \rightarrow b$  and  $g: b \rightarrow c$  in a category  $\mathbb{C}$ . It will be sometimes convenient to indicate an arrow  $f: a \rightarrow b$  of  $\mathbb{C}$  as  $x \xrightarrow{f \in \mathbb{C}} y$  or also  $\xrightarrow{\in \mathbb{C}}$ , if names are immaterial. For  $\mathbb{C}$  an SMC,  $\oplus$  is its monoidal product, with unit object  $I$ , and  $\sigma_{a,b}: a \oplus b \rightarrow b \oplus a$  is the symmetry associated with  $a, b \in \mathbb{C}$ . We write  $\bar{0}$  for  $\emptyset$  and  $\overline{n+1}$  for  $\{1, \dots, n, n+1\}$ .

## 2 PROPs

Our exposition is founded on PROPs (**product** and **permutation** categories [21]).

**Definition 2.1** A *PROP* is a symmetric strict monoidal category with objects the natural numbers, where  $\oplus$  on objects is addition. PROPs form a category **PROP** with morphisms the identity-on-objects symmetric strict monoidal functors.

A typical way of constructing a PROP is starting from a *symmetric monoidal theory* (SMT): it is a pair  $(\Sigma, E)$ , where  $\Sigma$  is a signature of *generators*  $o: n \rightarrow m$  with *arity*  $n$  and *coarity*  $m$ . The set of  $\Sigma$ -terms is obtained by composing generators in  $\Sigma$ , the unit  $id: 1 \rightarrow 1$  and the symmetry  $\sigma_{1,1}: 2 \rightarrow 2$  with  $;$  and  $\oplus$ . That means, given  $\Sigma$ -terms  $t: k \rightarrow l$ ,  $u: l \rightarrow m$ ,  $v: m \rightarrow n$ , one constructs new  $\Sigma$ -terms  $t; u: k \rightarrow m$  and  $t \oplus v: k + n \rightarrow l + n$ . The set  $E$  of *equations* contains pairs  $(t, t': n \rightarrow m)$  of  $\Sigma$ -terms with the same arity and coarity.

There is a natural graphical representation for  $\Sigma$ -terms using the formalism of string diagrams [25]. A  $\Sigma$ -term  $n \rightarrow m$  is pictured as a box with  $n$  ports on the left and  $m$  ports on the right. Composition  $t; s$  is rendered graphically as

$\boxed{\boxed{t} \boxed{s}}$  and  $t \oplus s$  as  $\boxed{\boxed{t} \oplus \boxed{s}}$ . The symmetric monoidal structure is generated from

$\boxed{\quad}$ , representing  $id_1: 1 \rightarrow 1$ ,  $\boxed{\quad}$ , representing  $id_0: 0 \rightarrow 0$ , and  $\boxed{\text{X}}$ , representing  $\sigma_{1,1}: 2 \rightarrow 2$ .

An SMT  $(\Sigma, E)$  *freely generates* a PROP  $\mathcal{T}$  by letting arrows  $n \rightarrow m$  in  $\mathcal{T}$  be  $\Sigma$ -terms modulo  $E$ . We say that  $(\Sigma, E)$  is a *presentation* of a PROP  $\mathcal{S}$  when  $\mathcal{S} \cong \mathcal{T}$ . When  $\Sigma' \subseteq \Sigma$  and  $E' \subseteq E$ , there is an evident inclusion PROP morphism from the PROP  $\mathcal{T}'$  generated by  $(\Sigma', E')$  to the one  $\mathcal{T}$  generated by  $(\Sigma, E)$ , for which henceforth we reserve notation  $\mathcal{T}' \hookrightarrow \mathcal{T}$ .

<sup>2</sup> This observation is also relevant for algebraic approaches to quantum processes, see e.g. [16, Th. 5.6].

**Example 2.2**

- (a) In the SMT  $(\Sigma_M, E_M)$  of *commutative monoids*,  $\Sigma_M$  contains a multiplication  $\boxed{\bullet} : 2 \rightarrow 1$  and a unit  $\boxed{\bullet} : 0 \rightarrow 1$ . Equations  $E_M$  assert associativity (M1), commutativity (M2) and unitality (M3).

$$\begin{array}{c} \text{Diagram 1} \end{array} = \begin{array}{c} \text{Diagram 2} \end{array} \quad (\text{M1}) \qquad \begin{array}{c} \text{Diagram 3} \end{array} = \begin{array}{c} \text{Diagram 4} \end{array} \quad (\text{M2}) \qquad \begin{array}{c} \text{Diagram 5} \end{array} = \begin{array}{c} \text{Diagram 6} \end{array} \quad (\text{M3})$$

$(\Sigma_M, E_M)$  presents the PROP  $\mathbf{F}$  whose arrows  $n \rightarrow m$  are total functions from  $\bar{n}$  to  $\bar{m}$ , with  $\bar{n} = \{1, \dots, n\}$ . Writing  $\mathbf{Mn}$  for the PROP freely generated by  $(\Sigma_M, E_M)$ , the isomorphism  $\mathbf{Mn} \cong \mathbf{F}$  is defined by interpreting string diagrams as graphs of functions. For instance, the diagram on the right represents the function  $\bar{3} \rightarrow \bar{3}$  mapping 1 on the left to 2 on the right and 2, 3 on the left to 1 on the right.

- (b) The SMT  $(\Sigma_C, E_C)$  of *cocommutative comonoids* is based on a comultiplication  $\boxed{\bullet} : 1 \rightarrow 2$  and a counit  $\boxed{\bullet} : 1 \rightarrow 0$ .  $E_C$  is the following set of equations.

$$\begin{array}{c} \text{Diagram 1} \end{array} = \begin{array}{c} \text{Diagram 2} \end{array} \quad (\text{C1}) \qquad \begin{array}{c} \text{Diagram 3} \end{array} = \begin{array}{c} \text{Diagram 4} \end{array} \quad (\text{C2}) \qquad \begin{array}{c} \text{Diagram 5} \end{array} = \begin{array}{c} \text{Diagram 6} \end{array} \quad (\text{C3})$$

We write  $\mathbf{Cm}$  for the PROP freely generated by  $(\Sigma_C, E_C)$ . There is an evident isomorphism  $\mathbf{Cm} \cong \mathbf{Mn}^{op}$  given by “vertical rotation” of string diagrams. Therefore,  $(\Sigma_C, E_C)$  presents  $\mathbf{F}^{op}$ .

- (c) The PROP  $\mathbf{Fr}$  of *special Frobenius algebras* [9] is generated by the theory  $(\Sigma_M \uplus \Sigma_C, E_M \uplus E_C \uplus F)$ , where  $F$  is the following set of equations.

$$\begin{array}{c} \text{Diagram 1} \end{array} = \begin{array}{c} \text{Diagram 2} \end{array} = \begin{array}{c} \text{Diagram 3} \end{array} \quad (\text{F1}) \qquad \begin{array}{c} \text{Diagram 4} \end{array} = \begin{array}{c} \text{Diagram 5} \end{array} \quad (\text{F2})$$

- (d) The PROP  $\mathbf{B}$  of (commutative/cocommutative) *bialgebras* is generated by the theory  $(\Sigma_M \uplus \Sigma_C, E_M \uplus E_C \uplus B)$ , where  $B$  is the following set of equations.

$$\begin{array}{c} \text{Diagram 1} \end{array} = \begin{array}{c} \text{Diagram 2} \end{array} \quad (\text{B1}) \qquad \begin{array}{c} \text{Diagram 3} \end{array} = \begin{array}{c} \text{Diagram 4} \end{array} \quad (\text{B3})$$

$$\begin{array}{c} \text{Diagram 5} \end{array} = \begin{array}{c} \text{Diagram 6} \end{array} \quad (\text{B2}) \qquad \begin{array}{c} \text{Diagram 7} \end{array} = \begin{array}{c} \text{Diagram 8} \end{array} \quad (\text{B4})$$

**Remark 2.3** The assertion that  $(\Sigma_M, E_M)$  is the SMT of *commutative monoids*—and similarly for other SMTs appearing in our exposition—can be made precise by establishing a correspondence between commutative monoids in a symmetric monoidal category  $\mathbb{C}$  and objects  $F(1)$  identified by symmetric monoidal functors  $F : \mathbf{Mn} \rightarrow \mathbb{C}$ , often called *models* or *algebras* of  $\mathbf{Mn}$ . As models are not central in our work, we refer the reader to [19] for more information.

### 3 PROP operations

The following table summarises three operations on given PROPs  $\mathcal{T}_1$  and  $\mathcal{T}_2$ . Supposing that they are presented by SMTs  $(\Sigma_1, E_1)$  and  $(\Sigma_2, E_2)$  respectively, the second column describes a presentation for the PROP resulting from the operation.

	PROPs	SMTs	Reference
<b>Sum</b>	$\mathcal{T}_1 + \mathcal{T}_2$	signature: $\Sigma_1 \uplus \Sigma_2$ equations: $E_1 \uplus E_2$	see e.g. [29, §2.3].
<b>Fibred sum over <math>\mathcal{T}_3</math></b>	$\mathcal{T}$ defined by $\begin{array}{ccc} \mathcal{T}_3 & \hookrightarrow & \mathcal{T}_1 \\ \downarrow & \lrcorner & \downarrow \\ \mathcal{T}_2 & \hookrightarrow & \mathcal{T} \end{array}$	sig.: $(\Sigma_1 \uplus \Sigma_2)_{\equiv_{\Sigma_3}}$ eq.: $(E_1 \uplus E_2)_{\equiv_{E_3}}$	see e.g. [29, §2.5].
<b>Composition via <math>\lambda</math></b>	$\mathcal{T}_1; \mathcal{T}_2$ defined by $\lambda: \mathcal{T}_2; \mathcal{T}_1 \rightarrow \mathcal{T}_1; \mathcal{T}_2.$	sig.: $\Sigma_1 \uplus \Sigma_2$ eq.: $E_1 \uplus E_2 \uplus E_\lambda$	introduced in [19], see also [29, §2.4].

We now illustrate the three operations. The simplest, the sum, just combines the two theories without adding any interaction.

The fibred sum mimics a kind of construction typical in algebra, from geometric gluing constructions of topological spaces to amalgamated free products of groups. The idea is to identify some structure  $\mathcal{T}_3$  that is in common between the two theories. In all applications, the assumption is that  $\Sigma_3 \subseteq \Sigma_1 \cap \Sigma_2$  and  $E_3 \subseteq E_1 \cap E_2$ : the quotient  $\equiv_{\Sigma_3}$  identifies  $o_1 \in \Sigma_1$  and  $o_2 \in \Sigma_2$  when  $o_1 = o_2$  is in  $\Sigma_3$ , and  $\equiv_{E_3}$  acts similarly on equations. On PROPs, this operation amounts to pushing out the inclusion morphisms  $\mathcal{T}_1 \hookrightarrow \mathcal{T}_3 \hookrightarrow \mathcal{T}_2$  from the PROP  $\mathcal{T}_3$  freely generated by  $(\Sigma_3, E_3)$ .

The composition enhances the sum with compatibility conditions between  $\mathcal{T}_1$  and  $\mathcal{T}_2$ . Also this operation mimics a standard pattern in algebra: e.g. a ring is given by a monoid and an abelian group, subject to equations that ensure that the former distributes over the latter. Formally, the operation  $\mathcal{T}_1; \mathcal{T}_2$  is defined in [19] by understanding PROPs  $\mathcal{T}_1, \mathcal{T}_2$  as monads in a certain bicategory [27], and then compose them via a distributive law  $\lambda: \mathcal{T}_2; \mathcal{T}_1 \rightarrow \mathcal{T}_1; \mathcal{T}_2$ . The resulting monad  $\mathcal{T}_1; \mathcal{T}_2$  is also a PROP, enjoying a presentation as the quotient of  $\mathcal{T}_1 + \mathcal{T}_2$  by the equations  $E_\lambda$  encoded by the distributive law. The set  $E_\lambda$  is simply the graph of  $\lambda$ , which can be seen as a set of directed equations  $(\xrightarrow{\in \mathcal{T}_2} \xrightarrow{\in \mathcal{T}_1}) \approx (\xrightarrow{\in \mathcal{T}_1} \xrightarrow{\in \mathcal{T}_2})$  telling how arrows of  $\mathcal{T}_2$  distribute over arrows of  $\mathcal{T}_1$ . In fortunate cases, like the examples below, it is possible to present  $E_\lambda$  by a simpler, or even finite, set of equations, thus giving a sensible axiomatisation of the compatibility conditions expressed by  $\lambda$ .

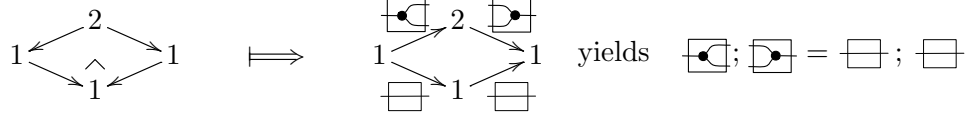
### Example 3.1

- (a) The PROP  $F$  of functions can be described as the composite  $Su; In$ , where  $Su$  and  $In$  are respectively the PROP of surjective and of injective functions [19]. The witnessing distributive law  $\lambda: In; Su \rightarrow Su; In$  maps a function  $\xrightarrow{\in In} \xrightarrow{\in Su}$  to its epi-mono factorisation  $\xrightarrow{\in Su} \xrightarrow{\in In}$ .

In more syntactic terms, using the isomorphism  $F \cong Mn$ , this result says that  $Mn$  is the composite  $Mu; Un$ , where  $Mu \cong Su$  is the PROP freely generated by the SMT  $(\{\boxed{\bullet}\}, \{(M1), (M2)\})$  and  $Un \cong In$  by the SMT  $(\{\boxed{\bullet}\}, \emptyset)$ . The distributive law explains the origin of equation (M3) of  $Mn$ , which indeed describes how to move the generator  $\boxed{\bullet}$  of  $Un$  past the one  $\boxed{\bullet}$  of  $Mu$ .

- (b) There is a distributive law  $\lambda: F^{op}; F \rightarrow F; F^{op}$  mapping a pair  $\xrightarrow{\in F^{op}} \xrightarrow{\in F}$ ,

i.e. a span  $\xleftarrow{\in F} \xrightarrow{\in F}$ , to (a choice of) its pushout  $\text{cospan} \xrightarrow{\in F} \xleftarrow{\in F}$ , i.e. a pair  $\xrightarrow{\in F} \xrightarrow{\in F^{op}}$  [19]. Because  $\mathbf{Mn} \cong \mathbf{F}$  and  $\mathbf{Cm} \cong \mathbf{F}^{op}$ , this yields a composite PROP  $\mathbf{Mn}; \mathbf{Cm}$ , presented as  $\mathbf{Mn} + \mathbf{Cm}$  modulo the equations arising from the distributive law. By definition of  $\lambda$ , such equations can be read from pushout squares in  $\mathbf{F}$ . For instance:



where the second diagram is obtained from the pullback by applying the isomorphisms  $\mathbf{F} \cong \mathbf{Mn}$  and  $\mathbf{F}^{op} \cong \mathbf{Cm}$ . In fact, Lack [19] shows that in order to present  $\lambda$  it suffices to check three pushout squares, corresponding to equations (F1)-(F2). Therefore,  $\mathbf{Mn}; \mathbf{Cm}$  is isomorphic to  $\mathbf{Fr}$  (Example 2.2), and both have a concrete description in terms of *cospan*s, i.e. the arrows of  $\mathbf{F}; \mathbf{F}^{op}$ .

- (c) Dually, there exists a distributive law  $\lambda: \mathbf{F}; \mathbf{F}^{op} \rightarrow \mathbf{F}^{op}; \mathbf{F}$ , defined by pullback in  $\mathbf{F}$  [19], which yields the PROP  $\mathbf{F}^{op}; \mathbf{F}$  of *spans*. All the equations arising by this distributive law can be proven from (B1)-(B4), yielding  $\mathbf{F}^{op}; \mathbf{F} \cong \mathbf{B}$ .

### Composing distributive laws

For our developments it is useful to generalise PROP composition to the case when there are more than two theories interacting with each other. The following result, a variation of a theorem by Cheng [10], is proven in [29, §2.4.6].

**Proposition 3.2** *Let  $\mathcal{F}$ ,  $\mathcal{H}$  and  $\mathcal{G}$  be PROPs presented by SMTs  $(\Sigma_{\mathcal{F}}, E_{\mathcal{F}})$ ,  $(\Sigma_{\mathcal{H}}, E_{\mathcal{H}})$  and  $(\Sigma_{\mathcal{G}}, E_{\mathcal{G}})$  respectively. Suppose there are distributive laws*

$$\lambda: \mathcal{H}; \mathcal{F} \rightarrow \mathcal{F}; \mathcal{H} \quad \chi: \mathcal{H}; \mathcal{G} \rightarrow \mathcal{G}; \mathcal{H} \quad \psi: \mathcal{G}; \mathcal{F} \rightarrow \mathcal{F}; \mathcal{G}$$

*satisfying the following “Yang-Baxter” equation:*

$$\begin{array}{c} \mathcal{H}; \mathcal{G}; \mathcal{F} \xrightarrow{\mathcal{H}\psi} \mathcal{H}; \mathcal{F}; \mathcal{G} \xrightarrow{\lambda\mathcal{G}} \mathcal{F}; \mathcal{H}; \mathcal{G} \xrightarrow{\mathcal{F}\chi} \mathcal{F}; \mathcal{G}; \mathcal{H} \\ \mathcal{H}; \mathcal{G}; \mathcal{F} \xrightarrow{\chi\mathcal{F}} \mathcal{G}; \mathcal{H}; \mathcal{F} \xrightarrow{\mathcal{G}\lambda} \mathcal{G}; \mathcal{F}; \mathcal{H} \xrightarrow{\psi\mathcal{H}} \mathcal{F}; \mathcal{G}; \mathcal{H} \end{array} \quad (3)$$

*then the following two are distributive laws:*

$$\left( \mathcal{H}; \mathcal{F}; \mathcal{G} \xrightarrow{\lambda\mathcal{G}} \mathcal{F}; \mathcal{H}; \mathcal{G} \xrightarrow{\mathcal{F}\chi} \mathcal{F}; \mathcal{G}; \mathcal{H} \right) \quad \left( \mathcal{G}; \mathcal{H}; \mathcal{F} \xrightarrow{\mathcal{G}\lambda} \mathcal{G}; \mathcal{F}; \mathcal{H} \xrightarrow{\psi\mathcal{H}} \mathcal{F}; \mathcal{G}; \mathcal{H} \right)$$

*yielding the same PROP  $\mathcal{F}; \mathcal{G}; \mathcal{H}$ . Furthermore, call  $E_{\lambda}$ ,  $E_{\chi}$  and  $E_{\psi}$  the sets of equations encoding the three laws. Then  $\mathcal{F}; \mathcal{G}; \mathcal{H}$  is presented by the signature  $\Sigma_{\mathcal{F}} \uplus \Sigma_{\mathcal{H}} \uplus \Sigma_{\mathcal{G}}$  and equations  $E_{\mathcal{F}} \uplus E_{\mathcal{H}} \uplus E_{\mathcal{G}} \uplus E_{\lambda} \uplus E_{\chi} \uplus E_{\psi}$ .*

**Example 3.3** We show how the PROP PF of *partial function* can be presented modularly using iterated distributive laws. First, we introduce a new PROP  $\mathbf{Cu}$ , generated by the signature  $\{\boxed{\bullet}\}$  and no equations: modulo the different colouring,

it is just  $\text{Un}^{op}$ . Following the recipe of Proposition 3.2, we now combine  $\text{Cu}$ ,  $\text{Un}$  and  $\text{Mu}$  via three distributive laws:


$$\lambda: \text{Un}; \text{Cu} \rightarrow \text{Cu}; \text{Un} \quad \chi: \text{Un}; \text{Mu} \rightarrow \text{Mu}; \text{Un} \quad \psi: \text{Mu}; \text{Cu} \rightarrow \text{Cu}; \text{Mu}$$

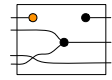
Using the isomorphisms  $\text{Un} \cong \text{In}$ ,  $\text{Cu} \cong \text{In}^{op}$  and  $\text{Mu} \cong \text{Su}$ , we can define  $\chi$  by epi-mono factorisation as in Example 3.1(a); therefore, the resulting PROP  $\text{Mu}; \text{Un}$  is  $\text{Mu} + \text{Un}$  quotiented by (M3). Because pullbacks in  $\mathbf{F}$  preserve both monos and epis, we define  $\lambda$  and  $\psi$  by pullback in  $\mathbf{F}$ . It is readily seen that  $\lambda$  and  $\psi$  are presented, respectively, by the first and the second equation below:

$$\begin{array}{|c|} \hline \bullet \text{---} \bullet \\ \hline \end{array} = \begin{array}{|c|} \hline \\ \hline \end{array} \quad (\text{P1}) \quad \begin{array}{|c|} \hline \bullet \text{---} \bullet \\ \hline \end{array} = \begin{array}{|c|} \hline \bullet \text{---} \bullet \\ \hline \end{array}. \quad (\text{P2})$$

Also,  $\lambda$ ,  $\chi$  and  $\psi$  verify the Yang-Baxter equation (3) and thus Proposition 3.2 yields a PROP  $\text{Cu}; \text{Mu}; \text{Un}$  presented as the quotient of  $\text{Cu} + \text{Mu} + \text{Un}$  by (M3), (P1) and (P2). By analogy with the total case  $\text{Mn} \cong \text{Mu}; \text{Un}$ , we shall use  $\text{PMn}$  (partial commutative monoids) as a shorthand for  $\text{Cu}; \text{Mu}; \text{Un}$ .

We now claim that  $\text{PMn} \cong \text{PF}$ . To see this, observe that partial functions  $n \xrightarrow{f \in \text{PF}} m$  are in bijective correspondence with spans  $n \xleftarrow{i \in \text{In}} z \xrightarrow{f \in \text{F}} m$ : the injection  $i$  tells on which elements  $\bar{z}$  of  $\bar{n}$  the function  $f$  is defined. Since  $\text{In}^{op} \cong \text{Cu}$  and  $\mathbf{F} \cong \text{Mn} \cong \text{Mu}; \text{Un}$ , this correspondence yields the desired isomorphism  $\text{PF} \cong \text{In}^{op}; \mathbf{F} \cong \text{Cu}; \text{Mu}; \text{Un} \cong \text{PMn}$ .

As a last remark, note that the factorisation property of  $\text{PMn}$  allows to interpret any arrow of this PROP as the graph of a partial function, where  indicates partiality. For instance, the diagram on the right represents the function  $\bar{4} \rightarrow \bar{3}$  undefined on 1 and mapping 2, 4 to 2 and 3 to 3.



## 4 A presentation of equivalence relations

This section builds modularly a presentation for the PROP  $\text{ER}$  of equivalence relations, using the operations introduced in § 3. In defining  $\text{ER}$ , we use the following notation:  $[e]$  is the symmetric and transitive closure of a relation  $e$  and  $d|_Y$  is the restriction of an equivalence relation  $d$  on a set  $X$  to a subset  $Y \subseteq X$ .

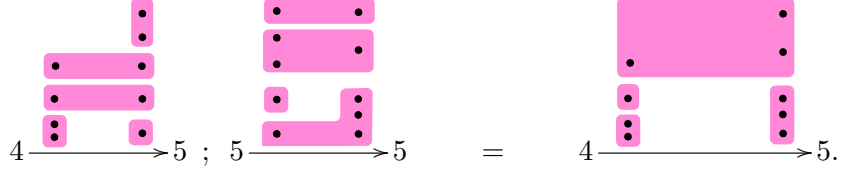
**Definition 4.1** Let  $\text{ER}$  be the PROP whose arrows  $n \rightarrow m$  are the equivalence relations on  $\bar{n} \uplus \bar{m}$ . Given  $e_1: n \rightarrow z$  and  $e_2: z \rightarrow m$ , the composite  $e_1; e_2: n \rightarrow m$  is defined in steps as follows.

$$\begin{aligned} e_1 * e_2 &:= \{(v, w) \mid \exists u. (v, u) \in e_1 \wedge (u, w) \in e_2\} \\ e_1 \diamond e_2 &:= e_1 \cup e_2 \cup [e_1 * e_2] \\ e_1; e_2 &:= e_1 \diamond e_2|_{\bar{n} \uplus \bar{m}} \end{aligned}$$

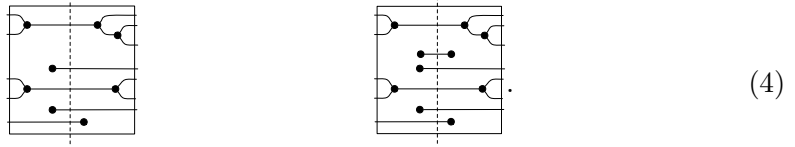
The monoidal product  $e_1 \oplus e_2$  is given by disjoint union of  $e_1$  and  $e_2$ .

In words, for composition one first defines an equivalence relation  $e_1 \diamond e_2$  on  $\bar{n} \uplus \bar{z} \uplus \bar{m}$  by gluing together equivalence classes of  $e_1$  and  $e_2$  along common witnesses

in  $\bar{z}$ , then obtains  $e_1; e_2$  by restricting to elements of  $\bar{n} \uplus \bar{m}$ . Here is an example:



Our approach in characterising ER stems from the observation that cospans can be interpreted as “redundant” equivalence relations. This becomes particularly neat when representing cospans as string diagrams via the characterisation  $\mathbf{Fr} \cong \mathbf{F}; \mathbf{F}^{op}$  (Example 3.1(b)), as below.



The dotted line emphasizes the fact that  $\mathbf{Fr}$  factorises as  $\mathbf{Mn}; \mathbf{Cm}$ . Both string diagrams in (4) define an equivalence relation  $e$  on  $\bar{5} \uplus \bar{7}$  by letting  $(v, w) \in e$  if the port associated with  $v$  and the one associated with  $w$  are linked in the graphical representation. For instance,  $1, 2 \in \bar{5}$  on the left boundary are in the same equivalence class as  $1, 2, 3 \in \bar{7}$  on the right boundary, whereas  $5 \in \bar{5}$  and  $4 \in \bar{7}$  are the only members of their equivalence class.

Observe that the two representations of  $e$  in (4) only differ for the sub-diagram  $\boxed{\bullet \bullet}$ , which indeed does not play any role in the interpretation and stands for an “empty” equivalence class. Equation (B4) will be employed to express the redundancy of  $\boxed{\bullet \bullet}$ . Let us call  $\mathbf{IFr}$  (irredundant **F**robenius algebras) the PROP defined as the quotient of  $\mathbf{Fr}$  by (B4). Our discussion leads to the following claim.

**Theorem 4.2**  $\mathbf{IFr} \cong \mathbf{ER}$ .

The isomorphism of Theorem 4.2 shall arise as a universal arrow in the following “cube” diagram in  $\mathbf{PROP}$ , provided that the top and bottom square are pushouts.

$$\begin{array}{ccccc}
 & & \text{Un} + \text{Cu} & \hookrightarrow & \text{Cu}; \text{Un} \\
 & \swarrow & \cong \downarrow & \searrow & \downarrow \cong \\
 \mathbf{Fr} & \hookrightarrow & & \rightarrow & \mathbf{IFr} \\
 \cong \downarrow & \swarrow [\iota_1, \iota_2] & \text{In} + \text{In}^{op} & \xrightarrow{[\kappa_1, \kappa_2]} & \text{In}^{op}; \text{In} \\
 \mathbf{F}; \mathbf{F}^{op} & \xrightarrow{\Pi} & & \rightarrow & \mathbf{ER} \\
 & & & \nwarrow \gamma & 
 \end{array} \tag{5}$$

First we explain the PROP morphisms in (5). Those of the top face are defined by inclusion of the corresponding SMTs and the rear vertical isomorphisms have been introduced in Examples 3.1-3.3. Thus we focus on the bottom face.

**Definition 4.3**

- morphisms  $\kappa_1: \text{In} \rightarrow \text{In}^{op}; \text{In}$ ,  $\kappa_2: \text{In}^{op} \rightarrow \text{In}^{op}; \text{In}$ ,  $\iota_1: \text{In} \rightarrow \mathbf{F}; \mathbf{F}^{op}$  and  $\iota_2: \text{In}^{op} \rightarrow \mathbf{F}; \mathbf{F}^{op}$  are given by

$$\begin{aligned}\kappa_1(n \xrightarrow{f} m) &= (n \xleftarrow{id} n \xrightarrow{f} m) & \kappa_2(n \xrightarrow{f} m) &= (n \xleftarrow{f} m \xrightarrow{id} m) \\ \iota_1(n \xrightarrow{f} m) &= (n \xrightarrow{f} m \xleftarrow{id} m) & \iota_2(n \xrightarrow{f} m) &= (n \xrightarrow{id} n \xleftarrow{f} m).\end{aligned}$$

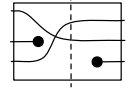
- $\Pi: \mathbf{F}; \mathbf{F}^{op} \rightarrow \mathbf{ER}$  is defined on a cospan  $n \xrightarrow{p} z \xleftarrow{q} m$  by



$$(v, w) \in \Pi(n \xleftarrow{f} z \xrightarrow{g} m) \quad \text{iff} \quad \begin{cases} p(v) = q(w) & \text{if } v \in \overline{n}, w \in \overline{m} \\ q(v) = p(w) & \text{if } v \in \overline{m}, w \in \overline{n} \\ p(v) = p(w) & \text{if } v, w \in \overline{n} \\ q(v) = q(w) & \text{if } v, w \in \overline{m}. \end{cases} \quad (6)$$

- $\Upsilon: \mathbf{In}^{op}; \mathbf{In} \rightarrow \mathbf{ER}$  is defined on a span  $n \xleftarrow{f \in \mathbf{In}} z \xrightarrow{g \in \mathbf{In}} m$  as the reflexive and symmetric closure of  $\{(v, w) \mid f^{-1}(v) = g^{-1}(w)\}$ .

It is lengthy but conceptually simple to verify that  $\Pi$  and  $\Upsilon$  are indeed functorial assignments — details are reported in [29, Appendix A].

Informally,  $\Pi$  implements the idea of interpreting a cospan as an equivalence relation. For  $\Upsilon$ , the key observation is that spans of injective functions can also be seen as equivalence relations. Once again, the graphical representation of an arrow of  $\mathbf{In}^{op}; \mathbf{In}$  as a string diagram in  $\mathbf{Cu}; \mathbf{Un}$  can help visualising this fact. A factorised arrow of  $\mathbf{Cu}; \mathbf{Un}$  as on the right can be interpreted as the equivalence relation associating 1 on the left boundary with 2 on the right boundary, 3 on the left with 1 on the right and letting 2 on the left, 3 on the right be the only representatives of their equivalence class.






Note that this interpretation would not work the same way for spans of non-injective functions, as their graphical representation in  $\mathbf{F}^{op}; \mathbf{F}$  may involve  and  — more on this in Remark 4.11.

As explained above, Theorem 4.2 will follow from the following two lemmas.

**Lemma 4.4** *The top face of (5) is a pushout.*

**Proof** The PROP  $\mathbf{Cu}; \mathbf{Un}$  is defined as in Example 3.3, by pullback in  $\mathbf{In}$ , whence it is presented as the quotient of  $\mathbf{Un} + \mathbf{Cu}$  by (B4). Therefore, by definition, the SMT of  $\mathbf{IFr}$  consists of the SMTs for  $\mathbf{Fr}$  and  $\mathbf{Cu}; \mathbf{Un}$ , modulo the identification of generators and equations of  $\mathbf{Un} + \mathbf{Cu}$ . This is the situation described by the fibered sum operation of § 3, which implies the statement of the lemma.  $\square$

**Lemma 4.5** *The bottom face of (5) is a pushout.*

We will get to the proof of Lemma 4.5 in steps. First, we need an understanding of when two cospans are identified by  $\Pi$ . (4) gives us a lead: two cospans represent the same equivalence relation precisely when they are the same modulo (B4). Now, since (B4) arises by a distributive law  $\mathbf{F}; \mathbf{F}^{op} \rightarrow \mathbf{F}^{op}; \mathbf{F}$  defined by pullback in  $\mathbf{F}$  (Example 3.1(b)), one could be tempted of claiming that  $\Pi$  identifies two cospans precisely when they have the same pullback. However, this approach identifies too much. A counterexample is given by cospans represented by  and , which have the same pullback  but express different partitions of  $\overline{2}$ . The correct

approach is subtler: since we only need to rewrite  $\boxed{\bullet \rightarrow \bullet}$  as  $\square$ , it suffices to pull back the region of the cospan where all sub-diagrams of shape  $\boxed{\bullet \rightarrow \bullet}$  lie. Formally, we decompose a cospan  $\xrightarrow{\in F} \xleftarrow{\in F}$  as  $\xrightarrow{\in Su} \xrightarrow{\in Ln} \xleftarrow{\in Ln} \xleftarrow{\in Su}$  using the factorisation  $F \cong Su ; Ln$  (Example 3.1(a)), and then pull back the middle cospan  $\xrightarrow{\in Ln} \xleftarrow{\in Ln}$ . This removes all sub-diagrams of shape  $\boxed{\bullet \rightarrow \bullet}$ , as in the following riproposition of (4).

(7)

We crystallise our approach with the following definition.

**Definition 4.6** We say that two cospans  $n \xrightarrow{p_1 \in F} z \xleftarrow{q_1 \in F} m$  and  $n \xrightarrow{p_2 \in F} r \xleftarrow{q_2 \in F} m$  are *equal modulo-zeros* if there is an epi-mono factorisation  $\xrightarrow{e_p^1 \in Su} \xrightarrow{m_p^1 \in Ln} \xleftarrow{m_q^1 \in Ln} \xleftarrow{e_q^1 \in Su}$  of  $p_1 \xrightarrow{q_1}$ , and one  $\xrightarrow{e_p^2 \in Su} \xrightarrow{m_p^2 \in Ln} \xleftarrow{m_q^2 \in Ln} \xleftarrow{e_q^2 \in Su}$  of  $p_2 \xrightarrow{q_2}$  such that  $\xrightarrow{m_p^1} \xleftarrow{m_q^1}$  and  $\xrightarrow{m_p^2} \xleftarrow{m_q^2}$  have the same pullback and  $e_p^1 = e_p^2$ ,  $e_q^1 = e_q^2$ .

**Remark 4.7** It may be insightful to remark that two cospans are equal modulo-zeros precisely when they are in the equivalence relation generated by

$$\left( n \xrightarrow{p} z \xleftarrow{q} m \right) \sim \left( n \xrightarrow{p} z \xrightarrow{h} z' \xleftarrow{h} z \xleftarrow{q} m \right), \text{ where } h \text{ is an injection.}$$

The idea is that  $z \xrightarrow{h} z' \xleftarrow{h} z$  plays a role akin to a repeated use of equation (B4) in the diagrammatic language: it deflates the codomain of  $[p, q]: n + m \rightarrow z$  so as to “make it surjective”.

Our proof of Lemma 4.5 relies on showing that  $\Pi$  equalizes two cospans precisely when they are equal modulo-zeros. As a preliminary step, we need to establish some properties holding for any  $\Gamma$ ,  $\Delta$  and  $\mathbb{X}$  making the following diagram commute.

(8)

**Lemma 4.8** Given a PROP  $\mathbb{X}$  and a commutative diagram (8), the following hold.

- (i) If  $\xrightarrow{p} \xleftarrow{q}$  is a cospan in  $Ln$  with pullback (in  $Ln$ )  $\xleftarrow{f} \xrightarrow{g}$ , then  $\Gamma(\xleftarrow{f} \xrightarrow{g}) = \Delta(\xrightarrow{p} \xleftarrow{q})$ .
- (ii) If  $\xleftarrow{p_1} \xrightarrow{q_1}$  and  $\xleftarrow{p_2} \xrightarrow{q_2}$  are cospans in  $Ln$  with the same pullback then  $\Delta(\xrightarrow{p_1} \xleftarrow{q_1}) = \Delta(\xrightarrow{p_2} \xleftarrow{q_2})$ .
- (iii) If  $\xrightarrow{p_1} \xleftarrow{q_1}$  and  $\xrightarrow{p_2} \xleftarrow{q_2}$  are equal modulo-zeros then  $\Delta(\xrightarrow{p_1} \xleftarrow{q_1}) = \Delta(\xrightarrow{p_2} \xleftarrow{q_2})$ .
- (iv) If  $\xleftarrow{f} \xrightarrow{g}$  is a span in  $Ln$  with pushout (in  $F$ )  $\xrightarrow{p} \xleftarrow{q}$ , then  $\Gamma(\xleftarrow{f} \xrightarrow{g}) = \Delta(\xrightarrow{p} \xleftarrow{q})$ .

**Proof**

(i) We have that  $\Delta(\xrightarrow{p} \xleftarrow{q}) = \Delta(\iota_1 p ; \iota_2 q) = \Delta \iota_1 p ; \Delta \iota_2 q = \Gamma \kappa_1 p ; \Gamma \kappa_2 q = \Gamma(\kappa_1 p ; \kappa_2 q) = \Gamma(\xleftarrow{f} \xrightarrow{g})$ .



- (ii) Let  $\xleftarrow{f} \xrightarrow{g}$  be the pullback of both  $\xrightarrow{p_1} \xleftarrow{q_1}$  and  $\xrightarrow{p_2} \xleftarrow{q_2}$ . By (i)  $\Gamma(\xleftarrow{f} \xrightarrow{g}) = \Delta(\xrightarrow{p_1} \xleftarrow{q_1})$  and  $\Gamma(\xleftarrow{f} \xrightarrow{g}) = \Delta(\xrightarrow{p_2} \xleftarrow{q_2})$ . The statement follows.
- (iii) By assumption  $n \xrightarrow{p_1} z \xleftarrow{q_1} m$  and  $n \xrightarrow{p_2} r \xleftarrow{q_2} m$  have epi-mono factorisations  $n \xrightarrow{e_p} \xrightarrow{m_p^1} z \xleftarrow{m_q^1} \xleftarrow{e_q} m$  and  $n \xrightarrow{e_p} \xrightarrow{m_p^2} r \xleftarrow{m_q^2} \xleftarrow{e_q} m$  respectively, where  $\xrightarrow{m_p^1} \xleftarrow{m_q^1}$  and  $\xrightarrow{m_p^2} \xleftarrow{m_q^2}$  have the same pullback. Then:

$$\begin{aligned} \Delta(\xrightarrow{p_1} \xleftarrow{q_1}) &= \Delta(\xrightarrow{e_p} \xrightarrow{m_p^1} \xleftarrow{m_q^1} \xleftarrow{e_q}) = \Delta(\xrightarrow{e_p} \xleftarrow{id}); \Delta(\xrightarrow{m_p^1} \xleftarrow{m_q^1}); \Delta(\xrightarrow{id} \xleftarrow{e_q}) \\ &\stackrel{(ii)}{=} \Delta(\xrightarrow{e_p^2} \xleftarrow{id}); \Delta(\xrightarrow{m_p^2} \xleftarrow{m_q^2}); \Delta(\xrightarrow{id} \xleftarrow{e_q}) = \Delta(\xrightarrow{e_p} \xrightarrow{m_p^2} \xleftarrow{m_q^2} \xleftarrow{e_q}) = \Delta(\xrightarrow{p_2} \xleftarrow{q_2}). \end{aligned}$$

- (iv) Analogous to (i).  $\square$

Lemma 4.8 states that any commutative diagram (8) equalizes all cospans that are equal modulo-zeros. In our cube (5), also the converse statement holds.

**Lemma 4.9** *The following are equivalent*

- (a)  $n \xrightarrow{p_1} z \xleftarrow{q_1} m$  and  $n \xrightarrow{p_2} r \xleftarrow{q_2} m$  are equal modulo zeros.  
 (b)  $\Pi(\xrightarrow{p_1} \xleftarrow{q_1}) = \Pi(\xrightarrow{p_2} \xleftarrow{q_2})$ .

**Proof** Since bottom face of (5) commutes (see Lemma A.1 in the Appendix), Lemma 4.8 yield the direction (a)  $\Rightarrow$  (b). For the converse direction, a routine check shows that the definition of  $\Pi$  enforces the two cospans to have epi-mono factorisations with the desired properties. For details, see Appendix A.  $\square$

We now have all the ingredients to show that the bottom face of (5) is a pushout.

**Proof of Lemma 4.5** Commutativity is given by Lemma A.1, thus it remains to show the universal property. Suppose that we have a commutative diagram as in (8). It suffices to show that there exists a PROP morphism  $\Theta: \text{ER} \rightarrow \mathbb{X}$  with  $\Theta\Upsilon = \Gamma$  and  $\Theta\Pi = \Delta$  – uniqueness is automatic by fullness of  $\Pi$  (Lemma A.2).

Given an equivalence relation  $e: n \rightarrow m$ , there exist a cospan  $\xrightarrow{p} \xleftarrow{q}$  such that  $\Pi(\xrightarrow{p} \xleftarrow{q}) = e$ . We let  $\Theta(e) = \Delta(\xrightarrow{p} \xleftarrow{q})$ . This is well-defined: if  $\xrightarrow{p'} \xleftarrow{q'}$  is another cospan such that  $\Pi(\xrightarrow{p'} \xleftarrow{q'}) = e$  then Lemma 4.9 says that  $\xrightarrow{p} \xleftarrow{q}$  and  $\xrightarrow{p'} \xleftarrow{q'}$  are equal modulo-zeros and thus, by Lemma 4.8,  $\Delta(\xrightarrow{p} \xleftarrow{q}) = \Delta(\xrightarrow{p'} \xleftarrow{q'})$ . This argument also shows that, generally,  $\Theta\Pi = \Delta$ . Finally,  $\Theta$  preserves composition:

$$\begin{aligned} \Theta(e; e') &= \Theta(\Pi(\xrightarrow{p} \xleftarrow{q}); \Pi(\xrightarrow{p'} \xleftarrow{q'})) = \Theta(\Pi((\xrightarrow{p} \xleftarrow{q}); (\xrightarrow{p'} \xleftarrow{q'}))) \\ &= \Delta((\xrightarrow{p} \xleftarrow{q}); (\xrightarrow{p'} \xleftarrow{q'})) = \Delta(\xrightarrow{p} \xleftarrow{q}); \Delta(\xrightarrow{p'} \xleftarrow{q'}) = \Theta(e); \Theta(e'). \end{aligned}$$

We conclude by showing  $\Theta\Upsilon = \Gamma$ : given a span  $\xleftarrow{f} \xrightarrow{g}$  in  $\text{In}$ , let  $\xrightarrow{p} \xleftarrow{q}$  be its pushout span in  $\text{F}$ . By Lemma 4.8.(iv),  $\Gamma(\xleftarrow{f} \xrightarrow{g}) = \Delta(\xrightarrow{p} \xleftarrow{q}) = \Theta\Pi(\xrightarrow{p} \xleftarrow{q}) = \Theta\Upsilon(\xleftarrow{f} \xrightarrow{g})$ .  $\square$

We can now conclude the characterisation of  $\text{ER}$ .

**Proof of Theorem 4.2** The top and the bottom face of (5) are pushouts by Lemma 4.4 and 4.5. This yields a unique PROP morphism  $\text{IFr} \rightarrow \text{ER}$  making the

diagram commute. Since the other vertical arrows in (5) are isomorphisms, then  $\text{IFr} \rightarrow \text{ER}$  is also an isomorphism.  $\square$

**Remark 4.10** As hinted by the rightmost diagram in (7), one can give an alternative characterisation of  $\text{ER}$  as the composite  $\text{PROP } \text{Su}; \text{In}^{op}; \text{In}; \text{Su}$ . This would rely on defining the appropriate distributive laws and combine them using Proposition 3.2: the resulting equations are precisely those of  $\text{IFr}$ . Then, showing that factorised arrows of  $\text{Su}; \text{In}^{op}; \text{In}; \text{Su}$  are in bijective correspondence with equivalence relations in  $\text{ER}$  completes the proof that  $\text{IFr} \cong \text{ER}$ . In our exposition we preferred to use the “cube” construction (5), as it applies also to linear and partial functions (cf. § 6). Also, it yields the isomorphism  $\text{IFr} \cong \text{ER}$  as a universal arrow.

**Remark 4.11** Our construction merges the theory of cospans of functions and of spans of injective functions to form the theory of equivalence relations. One may wonder what happens with a more symmetric approach, namely if we consider spans of arbitrary functions. Mimicking the cube construction (5) would result in the following diagram in  $\mathbf{PROP}$ , where the top and the bottom face are pushouts.

$$\begin{array}{ccccc}
 & \text{Mn} + \text{Cm} & \xrightarrow{\cong} & \text{B} & \\
 \text{Fr} & \xrightarrow{\quad} & \mathcal{T} & \xrightarrow{\quad} & \text{B} \\
 \cong \downarrow & & \downarrow & & \downarrow \cong \\
 & \text{F} + \text{F}^{op} & \xrightarrow{\quad} & \text{F}^{op}; \text{F} & \\
 \text{F}; \text{F}^{op} & \xrightarrow{\quad} & \cdot & \xrightarrow{\quad} & \cdot
 \end{array} \tag{9}$$

The SMT for  $\mathcal{T}$  includes the SMTs for  $\text{Fr}$  and  $\text{B}$ , allowing us to prove

$$\boxed{\quad} \stackrel{(\text{M3}), (\text{C3})}{=} \boxed{\text{---} \bullet \text{---} \bullet \text{---}} \stackrel{(\text{F1})}{=} \boxed{\text{---} \bullet \text{---} \bullet \text{---}} \stackrel{(\text{B2})}{=} \boxed{\text{---} \bullet \text{---} \bullet \text{---}} \stackrel{(\text{B1})}{=} \boxed{\text{---} \bullet \text{---} \bullet \text{---}} \stackrel{(\text{B4})}{=} \boxed{\text{---} \bullet \text{---} \bullet \text{---}}.$$

This derivation trivialises the theory, as it implies that any two arrows of the same type are equal. Thus  $\mathcal{T}$ , as well as the pushout object of the bottom face in (9), is the *terminal object* in  $\mathbf{PROP}$ : for any  $\text{PROP } \mathbf{S}$  there is a unique morphism that maps any arrow  $n \xrightarrow{\in \mathbf{S}} m$  into the unique arrow with that source and target in  $\mathcal{T}$ .

## 5 A presentation of partial equivalence relations

Building on the results of the previous section, we shall now characterise the  $\text{PROP}$   $\text{PER}$  of *partial equivalence relations* (PERs) via another cube construction. In defining  $\text{PER}$ , we write  $\text{dom}(e)$  for the set  $Y \subseteq X$  of elements on which a partial equivalence relation  $e$  on  $X$  is defined. Also, we reuse the operation  $- \diamond -$  introduced in defining  $\text{ER}$  (Definition 4.1).

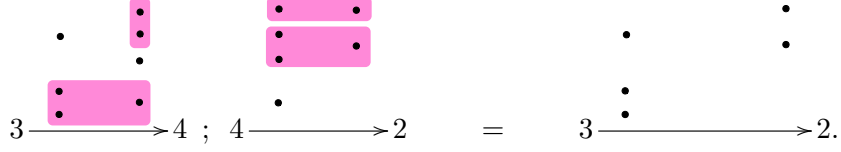
**Definition 5.1** Let  $\text{PER}$  be the  $\text{PROP}$  with arrows  $n \rightarrow m$  partial equivalence relations on  $\bar{n} \uplus \bar{m}$ . Given  $e_1: n \rightarrow z$ ,  $e_2: z \rightarrow m$ , the composite  $e_1; e_2$  is defined by

$$\begin{aligned}
 \Omega_{(e_1, e_2)} &:= \{u \in \bar{n} \uplus \bar{m} \mid \forall w \in \bar{z}. (u, w) \in e_1 \diamond e_2 \Rightarrow w \in \text{dom}(e_1) \cap \text{dom}(e_2)\} \\
 e_1; e_2 &:= e_1 \diamond e_2|_{\Omega_{(e_1, e_2)}}.
 \end{aligned}$$

The monoidal product  $e_1 \oplus e_2$  is given by disjoint union.

In words, composition in  $\text{PER}$  is defined as in  $\text{ER}$ , but  $e_1; e_2$  is left undefined on elements that, while gluing  $e_1$  and  $e_2$  into  $e_1 \diamond e_2$ , fall into the same equivalence

class as an element of  $\bar{z}$  on which either  $e_1$  or  $e_2$  is undefined. Here is an example in which the composite  $e_1 ; e_2$  turns out to be everywhere undefined:



We now discuss what SMT will present PER. As we did for equivalence relations, we first establish some preliminary intuition on the diagrammatic rendition of PERs. For functions, partiality was captured graphically by incorporating an additional generator  $\boxed{\bullet}$  (Example 3.3). The strategy for PERs is analogous: for the elements on which a PER  $e$  is defined, the diagrammatic description is the same given for equivalence relations in (4); the elements on which  $e$  is undefined will correspond instead to ports where we plug in  $\boxed{\bullet}$  (if on the left) or  $\boxed{\bullet}$  (if on the right).

Therefore, the string diagrammatic theory for PERs will involve  $\text{Fr}$  expanded with generators  $\boxed{\bullet}$ ,  $\boxed{\bullet}$ , subject to suitable compatibility conditions. This plan concretises into the PROP of “partial” special Frobenius algebras, whose definition relies on the PROP  $\text{PMn}$  discussed in Example 3.3.

**Definition 5.2** The PROP  $\text{PFr}$  is defined as  $\text{PMn} + \text{PMn}^{op}$  quotiented by equations (F1), (F2) and the following two.

$$\boxed{\bullet} = \boxed{\bullet} = \boxed{\bullet} \quad (\text{PFR1}) \qquad \boxed{\bullet} = \boxed{\bullet} \quad (\text{PFR2})$$

Intuitively, (PFR1) (together with (P1) and (P2) from  $\text{PMn}$  and their counterparts in  $\text{PMn}^{op}$ ) is the algebraic rendition of the “cancellation property” that we observed in the composition of partial equivalence relations.

As a partial version of  $\text{Fr}$ , we expect  $\text{PFr}$  to characterise cospans of partial functions. To phrase this statement, note that  $\text{PF}$  is equivalently described as the coslice category  $1/\mathbf{F}$  (that is, the skeletal category of pointed finite sets and functions) and thus has pushouts inherited from  $\mathbf{F}$ . We can then form the PROP  $\text{PF} ; \text{PF}^{op}$  of cospans in  $\text{PF}$  via a distributive law  $\text{PF}^{op} ; \text{PF} \rightarrow \text{PF} ; \text{PF}^{op}$  defined by pushout, analogously to the case of functions (Example 3.1(b)).

**Proposition 5.3**  $\text{PFr} \cong \text{PF} ; \text{PF}^{op}$ .

**Proof** For soundness of  $\text{PFr}$ , one simply needs to check that (PFR1) and (PFR2) can be read off pushout squares in  $\text{PF}$ , analogously to Example 3.1(c). Conversely, completeness amounts to show that any equation that can be read off pushout squares in  $\text{PF}$  is provable in  $\text{PFr}$ . The key insight is that any such pushout can be decomposed into simpler pushout squares only involving the generators of  $\text{PFr}$ . Thus it suffices to check that the interaction of generators is covered by the axioms of  $\text{PFr}$ . We leave further details for Appendix A.  $\square$

Now that we have an algebraic theory of cospans of partial functions, we can approach PERs by removing redundancy. Let us call  $\text{IPFr}$  (irredundant partial Frobenius algebras) the quotient of  $\text{PFr}$  by (B4).

**Theorem 5.4**  $\text{IPFr} \cong \text{PER}$ .

We proceed analogously to the case of equivalence relations. The isomorphism of Theorem 5.4 arises as a universal arrow in the following diagram in **PROP**, provided that the top and the bottom face are pushouts.

$$\begin{array}{ccccccc}
 & & \text{Un} + \text{Cu} & \xleftarrow{\quad} & \text{Fr} & \xleftarrow{\quad} & \text{PFr} \\
 & \swarrow & \downarrow \cong & \swarrow & \downarrow \cong & \swarrow & \downarrow \cong \\
 \text{Cu}; \text{Un} & \xleftarrow{\quad} & \text{IFr} & \xleftarrow{\quad} & \text{IPFr} & \xleftarrow{\quad} & \text{PFr} \\
 \downarrow \cong & & \downarrow \cong & & \downarrow \cong & & \downarrow \cong \\
 \text{In}^{op}; \text{In} & \xleftarrow{\quad} & \text{F}; \text{F}^{op} & \xleftarrow{\quad} & \text{PF}; \text{PF}^{op} & \xleftarrow{\quad} & \text{PF}; \text{PF}^{op} \\
 \downarrow \cong & & \downarrow \cong & & \downarrow \cong & & \downarrow \cong \\
 \text{In}^{op}; \text{In} & \xrightarrow{\quad} & \text{ER} & \xrightarrow{\quad} & \text{PER} & \xrightarrow{\quad} & \text{PER}
 \end{array} \quad (10)$$

The leftmost cube is just (5). We now specify  $\Lambda$ ,  $\Xi$  and  $\Pi'$ .

- For  $\Lambda$ , recall that there is a functor  $R: \text{PF} \rightarrow \text{F}$  which maps  $\bar{n}$  to  $\overline{n+1}$  and  $f: \bar{n} \rightarrow \bar{m}$  to the function  $\overline{n+1} \rightarrow \overline{m+1}$  sending to  $\star \in \bar{1}$  the elements on which  $f$  is undefined. Now,  $R$  has a left adjoint  $L: \text{F} \rightarrow \text{PF}$ : the obvious embedding of functions into partial functions. We define  $\Lambda$  as the embedding of  $\text{F}; \text{F}^{op}$  into  $\text{PF}; \text{PF}^{op}$  induced by  $L$ . This is a functorial assignment because left adjoints preserve pushouts.
- Similarly, we let  $\Xi$  be the obvious embedding of  $\text{ER}$  into  $\text{PER}$ . This assignment is functorial because composition in  $\text{PER}$  behaves as composition in  $\text{ER}$  on PERs that are totally defined.
- The PROP morphism  $\Pi': \text{PF}; \text{PF}^{op} \rightarrow \text{PER}$  is the extension of  $\Pi: \text{F}; \text{F}^{op} \rightarrow \text{ER}$  to partial functions, defined by the same clause (6). Note that the generality of  $\text{PER}$  is necessary: the value  $e$  of  $\Pi'$  on a cospan  $\xrightarrow{p} \xleftarrow{q}$  in  $\text{PF}$  is possibly not a reflexive relation, since  $p$  and  $q$  may be undefined on some elements of  $\bar{n}$ ,  $\bar{m}$ .

**Proof of Theorem 5.4** The leftmost top and bottom squares of (10) have been proven to be pushouts in Lemmas 4.4 and 4.5. The rightmost top square is readily seen to be a pushout by definition of the SMTs involved, similarly to the proof of Lemma 4.4. It thus remains to show that the rightmost bottom square is also a pushout. It clearly commutes by definition of  $\Pi$ ,  $\Pi'$ ,  $\Xi$  and  $\Lambda$ . To complete the proof, because  $\Lambda$  is an embedding, it suffices to check that  $\Xi(e) = \Pi'(\xrightarrow{p} \xleftarrow{q})$  precisely when there exist  $\xrightarrow{p'} \xleftarrow{q'}$  in  $\text{F}; \text{F}^{op}$  such that  $e = \Pi(\xrightarrow{p'} \xleftarrow{q'})$  and  $\Lambda(\xrightarrow{p'} \xleftarrow{q'}) = \xrightarrow{p} \xleftarrow{q}$ . We leave the (simple) details to Appendix A.

Finally, since the top and the bottom face of (10) are pushouts and the vertical arrows are isomorphisms, the universal arrow  $\text{IPFr} \rightarrow \text{PER}$  is also an isomorphism.  $\square$

## 6 Conclusions

Our work combines PROPs of spans and cospans of functions to give an algebraic characterisation for PROPs of equivalence relations. What we find most striking is that the same “cube” pattern leads to similar results in the total and partial cartesian case, explored here, and in the linear case, investigated in [5]. It seems that we are scratching the surface of a more general construction, which needs some

further insights to be better understood — as we saw, it collapses with spans of non-injective functions (Remark 4.11). We leave this investigation for future work.

**Acknowledgements.** Thanks to Filippo Bonchi, Pierre-Louis Curien, Peter Selinger, Pawel Sobociński and the anonymous referees for useful comments and discussion. The author acknowledges support from the ERC under the European Union’s Seventh Framework Programme (FP7/2007-2013) / ERC grant n° 320571.

## References

- [1] Baez, J. C. and J. Erbele, *Categories in control*, CoRR **abs/1405.6881** (2014).
- [2] Bonchi, F., P. Sobociński and F. Zanasi, *A categorical semantics of signal flow graphs*, in: *Concurrency Theory - 25th International Conference, CONCUR 2014. Proceedings*, 2014, pp. 435–450.
- [3] Bonchi, F., P. Sobociński and F. Zanasi, *Interacting bialgebras are Frobenius*, in: *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014*, 2014, pp. 351–365.
- [4] Bonchi, F., P. Sobociński and F. Zanasi, *Full abstraction for signal flow graphs*, in: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015*, 2015, pp. 515–526.
- [5] Bonchi, F., P. Sobociński and F. Zanasi, *Interacting Hopf algebras*, Journal of Pure and Applied Algebra (2016), to appear.  
URL <http://arxiv.org/abs/1403.7048>
- [6] Bruni, R. and F. Gadducci, *Some algebraic laws for spans (and their connections with multi-relations)*, in: *RelMiS 2001* (2001).
- [7] Bruni, R., H. C. Melgratti and U. Montanari, *A connector algebra for P/T nets interactions*, in: *CONCUR ‘11* (2011), pp. 312–326.
- [8] Bruni, R., U. Montanari, G. D. Plotkin and D. Terreni, *On hierarchical graphs: Reconciling bigraphs, gs-monoidal theories and gs-graphs*, Fundam. Inform. **134** (2014), pp. 287–317.
- [9] Carboni, A. and R. F. C. Walters, *Cartesian bicategories I*, J Pure Appl Algebra **49** (1987), pp. 11–32.
- [10] Cheng, E., *Iterated distributive laws.*, Math. Proc. Camb. Philos. Soc. **150** (2011), pp. 459–487.
- [11] Coecke, B. and R. Duncan, *Interacting quantum observables: categorical algebra and diagrammatics*, New Journal of Physics **13** (2011), p. 043016.
- [12] Coya, B. and B. Fong, *Corelations are the prop for extraspecial commutative Frobenius monoids*, CoRR **abs/1601.02307** (2016).
- [13] Došen, K. and Z. Petri, *Syntax for split preorders*, Annals of Pure and Applied Logic **164** (2013), pp. 443–481.
- [14] Duncan, R. and K. Dunne, *Interacting Frobenius algebras are Hopf*, CoRR **abs/1601.04964** (2016).
- [15] Hasuo, I. and N. Hoshino, *Semantics of higher-order quantum computation via geometry of interaction*, in: *Proceedings of the 26th IEEE Symposium on Logic in Computer Science, LICS*, 2011, pp. 237–246.
- [16] Heunen, C. and J. Vicary, *Lectures on categorical quantum mechanics* (2012).  
URL [www.cs.ox.ac.uk/files/4551/cqm-notes.pdf](http://www.cs.ox.ac.uk/files/4551/cqm-notes.pdf)
- [17] Jacobs, B., “Categorical Logic and Type Theory,” Number 141 in Studies in Logic and the Foundations of Mathematics, North Holland, Amsterdam, 1999.
- [18] Jacobs, B. and J. Mandemaker, *Coreflections in algebraic quantum logic*, Foundations of Physics **42** (2012), pp. 932–958.
- [19] Lack, S., *Composing PROPs*, Theor App Categories **13** (2004), pp. 147–163.
- [20] Lafont, Y., *Equational reasoning with 2-dimensional diagrams*, in: H. Comon and J.-P. Jounnaud, editors, *Term Rewriting*, Lecture Notes in Computer Science **909**, Springer Berlin Heidelberg, 1995 pp. 170–195.
- [21] Mac Lane, S., *Categorical algebra*, B Am Math Soc **71** (1965), pp. 40–106.

- [22] Mac Lane, S., “Categories for the Working Mathematician,” Springer, 1998.
- [23] Rosebrugh, R., N. Sabadini and R. Walters, *Generic commutative separable algebras and cospans of graphs*, Theory and applications of categories **15** (2005), pp. 164–177.
- [24] Scott, D., *Data types as lattices*, in: G. H. Müller, A. Oberschelp and K. Pothhoff, editors, *Proceedings of the International Summer Institute and Logic Colloquium, Kiel 1974* (1975), pp. 579–651.
- [25] Selinger, P., *A survey of graphical languages for monoidal categories*, Springer Lecture Notes in Physics **13** (2011), pp. 289–355.
- [26] Sobociński, P., *Nets, relations and linking diagrams*, in: *Algebra and Coalgebra in Computer Science - 5th International Conference, CALCO 2013*, 2013, pp. 282–298.
- [27] Street, R., *The formal theory of monads*, J Pure Appl Algebra **2** (2002), pp. 243–265.
- [28] Streicher, T., “Semantics of type theory : correctness, completeness, and independence results,” Progress in theoretical computer science, Birkhäuser, Boston, 1991.
- [29] Zanasi, F., “Interacting Hopf Algebras: the theory of linear systems,” Ph.D. thesis, Ecole Normale Supérieure de Lyon (2015).

## A Omitted Proofs

The following lemma is used in § 4.

**Lemma A.1** *The bottom face of (5) commutes.*

**Proof** It suffices to show that it commutes on the two injections into  $\mathbf{In} + \mathbf{In}^{op}$ , that means, for any  $f: n \rightarrow m$  in  $\mathbf{In}$ ,  $\Upsilon(\xleftarrow{id} \xrightarrow{f}) = \Pi(\xrightarrow{f} \xleftarrow{id})$  and  $\Upsilon(\xleftarrow{f} \xrightarrow{id}) = \Pi(\xrightarrow{id} \xleftarrow{f})$ . These statements are clearly symmetric, so it is enough to check one:

$$\begin{aligned} \Upsilon(\xleftarrow{id} \xrightarrow{f}) &= \{(v, w) \mid v = f^{-1}(w) \vee w = f^{-1}(v) \vee v = w\} \\ &= \{(v, w) \mid f(v) = w \vee f(w) = v \vee v = w\} \\ &\stackrel{f \in \mathbf{In}}{=} \{(v, w) \mid f(v) = w \vee f(w) = v \vee f(v) = f(w)\} = \Pi(\xrightarrow{f} \xleftarrow{id}). \end{aligned}$$

□

**Proof of Lemma 4.9** We complete the proof in the main text by showing that (b)  $\Rightarrow$  (a). For this purpose, it is useful to first verify the following properties:

- (i) for all  $u, u' \in \bar{n}$ ,  $p_1(u) = p_1(u')$  if and only if  $p_2(u) = p_2(u')$
- (ii) for all  $v, v' \in \bar{m}$ ,  $q_1(v) = q_1(v')$  if and only if  $q_2(v) = q_2(v')$
- (iii) for all  $u \in \bar{n}$ ,  $v \in \bar{m}$ ,  $p_1(u) = q_1(v)$  if and only if  $p_2(u) = q_2(v)$
- (iv) Let  $p_1[\bar{n}]$  be the number of elements of  $\bar{n}$  that are in the image of  $p_1$ , and similarly for  $p_2[\bar{n}]$ . Then  $p_1[\bar{n}] = p_2[\bar{n}]$ .
- (v)  $q_1[\bar{n}] = q_2[\bar{n}]$ .

For statement (i), observe that, by definition of  $\Pi$ , for any two elements  $u, u' \in \bar{n}$  the pair  $(u, u')$  is in  $\Pi(\xrightarrow{p_1} \xleftarrow{q_1})$  if and only if  $p_1(u) = p_1(u')$ . Similarly,  $(u, u') \in \Pi(\xrightarrow{p_2} \xleftarrow{q_2})$  if and only if  $p_2(u) = p_2(u')$ . Since by assumption  $\Pi(\xrightarrow{p_1} \xleftarrow{q_1}) = \Pi(\xrightarrow{p_2} \xleftarrow{q_2})$ , we obtain (i). A symmetric reasoning yields (ii). The argument for statement (iii) is analogous: for  $i \in \{1, 2\}$  and  $u \in \bar{n}$ ,  $v \in \bar{m}$ , by definition of  $\Pi$ ,  $(u, v) \in \Pi(\xrightarrow{p_i} \xleftarrow{q_i})$  if and only if  $p_i(u) = q_i(v)$ . Since  $\Pi(\xrightarrow{p_1} \xleftarrow{q_1}) = \Pi(\xrightarrow{p_2} \xleftarrow{q_2})$ , we obtain (iii). Statement (iv) is an immediate consequence of (i), and (v) of (ii).

Now, by virtue of properties (i)-(v), it should be clear that we can define epi-mono factorisations  $n \xrightarrow{e_p^1} \xrightarrow{m_p^1} z \xleftarrow{m_q^1} \xleftarrow{e_q^1} m$  and  $n \xrightarrow{e_p^2} \xrightarrow{m_p^2} r \xleftarrow{m_q^2} \xleftarrow{e_q^2} m$  of  $n \xrightarrow{p_1} z \xleftarrow{q_1} m$  and  $n \xrightarrow{p_2} r \xleftarrow{q_2} m$  respectively, with the following properties.

- (vi)  $e_p^1$  and  $e_p^2$  are the same function, with source  $n$  and target  $p_1[\bar{n}] = p_2[\bar{n}]$ . Also  $e_q^1$  and  $e_q^2$  are the same function, with source  $m$  and target  $q_1[\bar{m}] = q_2[\bar{m}]$ .
- (vii) For all  $u \in p_1[\bar{n}] = p_2[\bar{n}]$  and  $v \in q_1[\bar{n}] = q_2[\bar{n}]$ ,  $m_p^1(u) = m_q^1(v)$  iff  $m_p^2(u) = m_q^2(v)$ .

It remains to prove that  $\xrightarrow{m_p^1} \xleftarrow{m_q^1}$  and  $\xrightarrow{m_p^2} \xleftarrow{m_q^2}$  have the same pullback. For this purpose, let the following be pullback squares in  $\mathbf{In}$ :

$$\begin{array}{ccc} h_1 & \xrightarrow{f_1} & q_1[\bar{n}] \\ g_1 \downarrow & & \downarrow m_q^1 \\ p_1[\bar{n}] & \xrightarrow{m_p^1} & z \end{array} \quad \begin{array}{ccc} h_2 & \xrightarrow{f_2} & q_1[\bar{n}] \\ g_2 \downarrow & & \downarrow m_q^2 \\ p_1[\bar{n}] & \xrightarrow{m_p^2} & r \end{array}$$

By the way pullbacks are computed in  $\text{In}$  (i.e., in  $\mathbf{F}$ ), using (vii) we can conclude that  $m_p^1 g_2 = m_q^1 f_2$  and  $m_p^2 g_1 = m_q^2 f_1$ . By universal property of pullbacks, this implies that the spans  $\overleftarrow{g_1} \xrightarrow{f_1}$  and  $\overleftarrow{g_2} \xrightarrow{f_2}$  are isomorphic.  $\square$

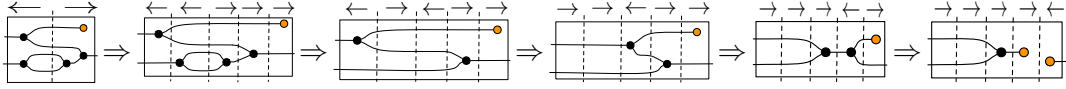
The following observation is used in the proof of Lemma 4.5.

**Lemma A.2**  $\Pi: \mathbf{F}; \mathbf{F}^{op} \rightarrow \mathbf{ER}$  is full.

**Proof** Let  $c_1, \dots, c_k$  be the equivalence classes of an equivalence relation  $e$  on  $\bar{n} \uplus \bar{m}$ . We define a cospan  $n \xrightarrow{p} k \xleftarrow{q} m$  by letting  $p$  map  $v \in \bar{n}$  to the equivalence class  $c_i$  to which  $v$  belongs, and symmetrically for  $q$  on values  $w \in \bar{m}$ . It is routine to check that  $\Pi(\xrightarrow{p} \xleftarrow{q}) = e$ .  $\square$

Next, we give more details on the proof of Proposition 5.3. The hard part is to check that the equation associated with any pushout diagram in  $\mathbf{PF}$  is provable by the equations of  $\mathbf{PFr}$ . The key observation is that we can confine ourselves to just pushouts involving the generators of  $\mathbf{PMn}$ .

Before making this formal, we illustrate the idea of the argument with the following example. The leftmost diagram below is a diagram representing a span  $\overleftarrow{f} \xrightarrow{g}$  (left), which we transform into a cospan (right) pushing out  $\overleftarrow{f} \xrightarrow{g}$ , only using equations of  $\mathbf{PFr}$ .



The steps are as follows. First, we expand  $\overleftarrow{f}$  and  $\overrightarrow{g}$  as  $\overleftarrow{f_1} \overleftarrow{f_2}$  and  $\overrightarrow{g_1} \overrightarrow{g_2} \overrightarrow{g_3}$  respectively, in such a way that each  $f_i$  and  $g_i$  contains at most one generator of  $\mathbf{PF}$  and  $\mathbf{PF}^{op}$ . In the next steps, we proceed pushing out spans  $\overleftarrow{f_i} \overrightarrow{g_j}$  whenever possible: graphically, this amounts to apply valid equations of  $\mathbf{PFr}$  of a very simple kind, namely those describing the interaction of a single (or no) generator of  $\mathbf{PF}^{op}$  with one (or none) of  $\mathbf{PF}$ . Note that pushing out spans of this form always gives back a cospan  $\xrightarrow{p} \xleftarrow{q}$  with  $p, q$  containing at most one generator, meaning that the procedure can be applied again until no more spans appear. The resulting diagram (the rightmost above) is the pushout of the leftmost one by pasting properties of pushouts. Therefore, we just proved that the equation

$$\overleftarrow{f} \xrightarrow{g} = \overrightarrow{g} \xleftarrow{f}$$

arising by the distributive law  $\mathbf{PF}^{op}; \mathbf{PF} \rightarrow \mathbf{PF}; \mathbf{PF}^{op}$  is provable in  $\mathbf{PFr}$ .

We now formalise the argument sketched above. Let us call *atom* any diagram of  $\mathbf{PMn}$  of shape  $\overleftarrow{f} \xrightarrow{b} \overrightarrow{g}$ , where  $f$  and  $g$  consist of components  $\square$  and  $\boxtimes$  composed together via  $\oplus$  or  $;$ , and  $b$  is either  $\square$  or a generator of  $\mathbf{PMn}$ . The following lemma establishes that  $\mathbf{PFr}$  is complete for pushouts involving atoms.

**Lemma A.3** Let  $\overleftarrow{f} \xrightarrow{g}$  be a span in  $\mathbf{PF}$  where  $f$  and  $g$  are in the image (under the isomorphism  $\mathbf{PMn} \cong \mathbf{PF}$ ) of atoms and suppose that the following is a pushout



square.

$$\begin{array}{ccc}
 & f & r & g \\
 m & \swarrow & & \searrow & n \\
 & p & z & q
 \end{array} \tag{A.1}$$

Then (i)  $p$  and  $q$  are also in the image of atoms and (ii) the associated equation is provable in  $\text{PFr}$ .

**Proof** The two points are proved by case analysis on all the possible choices of generators of  $\text{PMn}$  and  $(\text{PMn})^{op}$ .  $\square$

**Proof of Proposition 5.3** Fix any pushout square (A.1) in  $\text{PF}$  and pick expansions  $f = f_1 ; \dots ; f_k$  and  $g = g_1 ; \dots ; g_j$ , with each  $f_i$  and  $g_i$  in the image of an atom. We can calculate the pushout above by tiling pushouts of atoms as follows:

$$\begin{array}{ccccccc}
 & & f_2 & & f_1 & z & g_1 \\
 & & \swarrow & & \swarrow & & \searrow \\
 f_k & \dots & & & & & g_2 & \dots & g_j \\
 & & \searrow & & \searrow & & \swarrow \\
 & & & & & & & & 
 \end{array} \tag{A.2}$$

Point (i) of Lemma A.3 guarantees that each inner square only involves arrows in the image of some atom and Point (ii) ensures that all the associated equations are provable in  $\text{PFr}$ . It follows that also the equation associated with the outer pushout (A.2) is provable.  $\square$

We complete the proof sketch of Theorem 5.4 given in the main text. The following is the key lemma.

**Lemma A.4** Let  $e \in \text{ER}[n, m]$  and  $\frac{p}{\rightarrow} \leftarrow \frac{q}{\leftarrow} \in \text{PF} ; \text{PF}^{op}$ . The following are equivalent.

- (i)  $\Xi(e) = \Pi'(\frac{p}{\rightarrow} \leftarrow \frac{q}{\leftarrow})$ .
- (ii) There are cospans  $\frac{p_1}{\rightarrow} \leftarrow \frac{q_1}{\leftarrow}, \dots, \frac{p_k}{\rightarrow} \leftarrow \frac{q_k}{\leftarrow}$  in  $\text{F} ; \text{F}^{op}[n, m]$  such that

$$\begin{aligned}
 e &= \Pi(\frac{p_1}{\rightarrow} \leftarrow \frac{q_1}{\leftarrow}) \\
 \Lambda(\frac{p_1}{\rightarrow} \leftarrow \frac{q_1}{\leftarrow}) &= \Lambda(\frac{p_2}{\rightarrow} \leftarrow \frac{q_2}{\leftarrow}) \\
 \Pi(\frac{p_2}{\rightarrow} \leftarrow \frac{q_2}{\leftarrow}) &= \Pi(\frac{p_3}{\rightarrow} \leftarrow \frac{q_3}{\leftarrow}) \\
 &\dots\dots\dots \\
 \Lambda(\frac{p_k}{\rightarrow} \leftarrow \frac{q_k}{\leftarrow}) &= \frac{p}{\rightarrow} \leftarrow \frac{q}{\leftarrow} .
 \end{aligned}$$

**Proof** First we observe that, because  $\Lambda$  is an embedding,  $\Lambda(\frac{p_i}{\rightarrow} \leftarrow \frac{q_i}{\leftarrow}) = \Lambda(\frac{p_{i+1}}{\rightarrow} \leftarrow \frac{q_{i+1}}{\leftarrow})$  implies  $\frac{p_i}{\rightarrow} \leftarrow \frac{q_i}{\leftarrow} = \frac{p_{i+1}}{\rightarrow} \leftarrow \frac{q_{i+1}}{\leftarrow}$ . It follows that (ii) is equivalent to the statement that (iii) there exist  $\frac{p'}{\rightarrow} \leftarrow \frac{q'}{\leftarrow} \in \text{F} ; \text{F}^{op}[n, m]$  such that  $e = \Pi(\frac{p'}{\rightarrow} \leftarrow \frac{q'}{\leftarrow})$  and  $\Lambda(\frac{p'}{\rightarrow} \leftarrow \frac{q'}{\leftarrow}) = \frac{p}{\rightarrow} \leftarrow \frac{q}{\leftarrow}$ .

It is very easy to show that (iii) implies (i):

$$\Xi(e) \stackrel{(iii)}{=} \Xi\Pi(\frac{p'}{\rightarrow} \leftarrow \frac{q'}{\leftarrow}) \stackrel{\text{comm. of (10)}}{=} \Pi'\Lambda(\frac{p'}{\rightarrow} \leftarrow \frac{q'}{\leftarrow}) \stackrel{(iii)}{=} \Pi'(\frac{p}{\rightarrow} \leftarrow \frac{q}{\leftarrow}).$$

For the converse direction, suppose that we can show (\*) the existence of  $\frac{p'}{\rightarrow} \leftarrow \frac{q'}{\leftarrow} \in \text{F} ; \text{F}^{op}[n, m]$  such that  $\frac{p'}{\rightarrow} \leftarrow \frac{q'}{\leftarrow} = \Lambda(\frac{p}{\rightarrow} \leftarrow \frac{q}{\leftarrow})$ . Then the following derivation gives

statement (iii):

$$\Xi(e) \stackrel{(i)}{=} \Pi'(\xrightarrow[p]{q}) \stackrel{(*)}{=} \Pi'\Lambda(\xrightarrow[p']{q'}) \stackrel{\text{comm. of (10)}}{=} \Xi\Pi(\xrightarrow[p']{q'}).$$

Indeed, because  $\Xi$  is an embedding, the derivation above implies that  $e = \Pi(\xrightarrow[p']{q'})$ . Therefore it suffices to show  $(*)$ . For this purpose, we just need to prove that both  $n \xrightarrow{p \in \text{PF}} z$  and  $m \xrightarrow{q \in \text{PF}} z$  are *total* functions. Let  $u$  be an element of  $n$ : since  $\Pi'(\xrightarrow[p]{q}) = \Xi(e)$  and  $\Xi$  embeds equivalence relations into PERs, then  $\Pi'(\xrightarrow[p]{q})$  is in fact an equivalence relation, meaning that  $u$  belongs to some equivalence class of the partition induced by  $\Pi'(\xrightarrow[p]{q})$ . It follows by definition of  $\Pi'$  that  $p: n \rightarrow z$  is defined on  $u$ . With a similar argument, one can show that  $q: m \rightarrow z$  is defined on all elements of  $m$  and thus both  $p$  and  $q$  are total functions. This implies that  $\xrightarrow[p]{q}$  is in the image of the embedding  $\Lambda$ .  $\square$

**Proof of Theorem 5.4** In order to complete the proof of the main text, it remains to show that the leftmost bottom face of (10) is a pushout. First, recall that pushouts in **PROP** can be calculated as in **Cat**. In particular, (10) involves categories all with the same objects and identity-on-objects functors. This means that the pushout object is the quotient of **ER** and  $\mathbf{F}; \mathbf{F}^{op}$  along the equivalence relation generated by

$$\{(e, \xrightarrow[p]{q}) \mid \text{there is } \xrightarrow[p']{q'} \text{ such that } \Pi(\xrightarrow[p']{q'}) = e \text{ and } \Lambda(\xrightarrow[p']{q'}) = \xrightarrow[p]{q}\}. \quad (\text{A.3})$$

Lemma A.4 proves that  $\Pi'$  and  $\Xi$  map  $n \xrightarrow{e \in \text{ER}} m$  and  $n \xrightarrow[p]{q} m$  to the same arrow exactly when they are in the equivalence relation described above. This means that PER indeed quotients by (A.3) and thus is the desired pushout object.  $\square$

# How to think of intersection types as Cartesian products: marginalia to a theorem of Bucciarelli, Piperno, and Salvo

Rick Statman  
Carnegie Mellon University  
Department of Mathematical Sciences  
Pittsburgh, PA 15213  
statman@cs.cmu.edu

May 20, 2016

In their paper “Intersection types and lambda definability” [2] Bucciarelli, Piperno, and Salvo give a mapping of the strongly normalizable untyped terms into the simply typed terms via the assignment of intersection types. Here we shall both generalize their result and provide a converse. We shall do this by retracting untyped terms with surjective pairing onto untyped terms without pairing by using a special variant of Stovring’s notion [7] of a symmetric term. The symmetric ones which have simple types with Cartesian products are precisely the ones which retract onto (eta expansions of) strongly normalizable untyped terms. The intersection types of the strongly normalizable terms are related to the simple types with products in that we just replace  $\wedge$  by Cartesian product and vice versa.

Definition of the atoms of the language:  
the variables  $x, y, z, \dots$  are atoms  
the constants  $P, L, R$  are atoms

Definition of the terms of the language:  
atoms are terms  
if  $X, Y$  are terms then so are  $(XY)$  and  $\lambda x X$

We shall adopt the customary conventions:

- (i) parens are deleted and restored by left association and the use of Church's infix "dot" notation
- (ii) parens are added around abstractions, and additional unary operations, for readability.

The axiom and rules of untyped lambda calculus are the following. The first 5 axioms correspond to the classical theory of untyped lambda calculus with surjective pairing SP.

$$\begin{aligned}
 (\text{beta}) \quad (\lambda x X)Y &= [Y/x]X \\
 (\text{eta}) \quad X &= \lambda x. Xx \quad x \text{ not free in } X \\
 (L/Pa) \quad L(PXY) &= X \\
 (R/Pa) \quad R(PXY) &= Y \\
 (P/Dp) \quad P(LX)(RX) &= X
 \end{aligned}$$

The next 6 axioms correspond to the extended theory of Stovring (FP) and Statman (PSP [6], in the combinator case), which enjoys the Church-Rosser property when formulated by reductions.

$$\begin{aligned}
 (P/Ap) \quad PXYZ &= P(XZ)(YZ) \\
 (L/Ap) \quad LXY &= L(XY) \\
 (R/Ap) \quad RXY &= R(XY) \\
 (L/Ab) \quad L(\lambda x X) &= \lambda x(LX) \\
 (R/Ab) \quad R(\lambda x X) &= \lambda x(RX) \\
 (P/Ab) \quad P(\lambda x X)(\lambda x Y) &= \lambda x PXY
 \end{aligned}$$

There are certain useful derived rules.

- (1)  $(P/Dp)$  and  $(P/Ap) \Rightarrow (L/Ap)$  and  $(R/Ap)$   
 $L(XY) = L(P(LX)(RX)Y) = L(P(LXY)(RXY)) = LXY$   
 similarly for  $R$
- (2)  $(L/Ap)$  and  $(R/Ap)$  and  $(P/Dp) \Rightarrow (P/Ap)$   
 $L(PXYZ) = L(PXY)Z = XZ$  and  $R(PXYZ) = R(PXY)Z = YZ$  therefore  
 $PXYZ = P(L(PXYZ))(R(PXYZ))y = P(XZ)(YZ).$

(3) (eta) and  $(P/Ap) \Rightarrow (P/Ab)$

$$P(\lambda x X)(\lambda x Y) = \lambda y. P(\lambda x X)(\lambda x Y)y = \lambda y. P((\lambda x X)y)((\lambda x Y)y) = \lambda x PXY$$

(4) (eta) and  $(L/Ap) \Rightarrow (L/Ab)$

$$L(\lambda x X) = \lambda y. L(\lambda x X)y = \lambda y. L((\lambda x X)y) = \lambda x(LX)$$

similarly for  $R$

(5) (eta)  $\Rightarrow LP = K$  and  $RP = K^*$

$$L(PX) = \lambda x. L(PX)x = \lambda x. L(PXx) = \lambda x. X \text{ thus}$$

$$LP = \lambda x. LPx = \lambda x. L(Px) = \lambda xy. x = K$$

similarly

$$R(PX) = \lambda x. R(PX)x = \lambda x. R(PXx) = \lambda x. x \text{ hence}$$

$$RP = \lambda yx. x = K^*$$

The result of Klop is that the Church-Rosser property fails for the following classical reductions for  $SP$ :

$$(\text{beta}) \quad (\lambda x X)Y \Rightarrow [Y/x]X$$

$$(\text{eta}) \quad \lambda x. Xx \Rightarrow X \quad x \text{ not free in } X$$

$$(L/Pa) \quad L(PXY) \Rightarrow X$$

$$(R/Pa) \quad R(PXY) \Rightarrow Y$$

$$(P/Dp) \quad P(LX)(RX) \Rightarrow X$$

Nevertheless, this theory was proved conservative over beta-eta by de Vrijer [2]. Stovring and, later, Statman (for the combinator case) introduced new reductions for the first 5 and an additional one which enjoy Church-Rosser.

Stovring Reductions for FP with eta:

$$\begin{aligned}
(\text{beta}) \quad & (\lambda x X)Y \Rightarrow [Y/x]X \\
(\text{etae}) \quad & X \Rightarrow \lambda x. Xx \quad x \text{ not free in } X \\
(L/Pa) \quad & L(PXY) \Rightarrow X \\
(R/Pa) \quad & R(PXY) \Rightarrow Y \\
(P/De) \quad & X \Rightarrow P(LX)(RX) \\
(P/Ap) \quad & PXYZ \Rightarrow P(XZ)(YZ)
\end{aligned}$$

Now, Church-Rosser is enough to obtain de Vrijer's theorem and a co-de Vrijer theorem that beta-eta is conservative over pairing with FP. Stovring's argument uses the notion of symmetric term defined below.

Our basic theory of intersection types is Barendregt, Coppo, and Dezani (BCD, p. 580). There is a simpler theory consisting of  $\rightarrow I, \rightarrow E, \wedge I$ , and  $\wedge E$ . We shall call the later BPS since it is easy to see that is equivalent to the theory  $S$  of intersection types defined in [2]. Each intersection type  $A$  can be converted into a simple type  $A'$  with products by replacing  $(B \wedge C)$  with  $(B^*C)$ . The inverse of ' is ''.

Definition of simple types with Cartesian products:

Type atoms  $p, q, r, \dots$  are types.

If  $A$  and  $B$  are types then so are  $(A \rightarrow B)$  and  $(A^*B)$

We shall employ Curry's substitution prefix both for terms and types.  $[B/p]A$  is the result of substituting  $B$  for  $p$  in  $A$ . We adopt Church typing for terms as follows, but it is convenient to adopt the Curry notation  $S \vdash X : A$ , where  $S$  (the base) is a set of declarations  $x : B$  for the Church typings. Here  $S$  will only contain declarations  $x : B$  provided  $x$  is indeed a typed variable of type  $B$ .

Definition of typed terms:

$$\begin{aligned}
& x^A, y^B, z^C, \dots \text{ are typed variables} \\
& \vdash x^A : A, \vdash y^B : B, \vdash z^C : C, \dots
\end{aligned}$$

We abbreviate by omitting superscripts.

$$\begin{aligned}
\vdash X : A, \vdash Y : B &\Rightarrow \vdash PXY : (A^*B) \\
\vdash X : (A^*B) &\Rightarrow \vdash LX : A, \vdash RX : B \\
\vdash X : B &\Rightarrow \vdash \lambda x^A X : A \rightarrow B \\
\vdash X : A \rightarrow B, \vdash Y : A &\Rightarrow \vdash (XY) : B \\
\vdash X : A \rightarrow B, \vdash Y : A &\Rightarrow \vdash \langle XY \rangle : B \\
\vdash \langle (LX)Y \rangle : B, \vdash \langle (RX)Y \rangle : C &\Rightarrow \vdash \langle XY \rangle : B^*C
\end{aligned}$$

Here  $\langle \rangle$  simply represents the pointwise action of a pair on an element in the common domain of the coordinates. We could define  $\langle \rangle$  as follows:

$$\begin{aligned}
\vdash X : A \rightarrow B, \vdash Y : A &\rightarrow \langle XY \rangle := (XY) \\
\vdash \langle (LX)Y \rangle : B, \vdash \langle (RX)Y \rangle : C &\rightarrow \langle XY \rangle := P\langle (LX)Y \rangle \langle (RX)Y \rangle
\end{aligned}$$

but we prefer to introduce  $\langle \rangle$  as a primitive with reduction rules:

$$\begin{aligned}
(\text{beta})(\lambda x.XY) &\Rightarrow [Y/x]X \\
(L/Pa) L(PXY) &\Rightarrow X \\
(R/Pa) R(PXY) &\Rightarrow Y \\
(P/Dp) P(LX)(RX) &\Rightarrow X \\
(P/ <>) \langle (PXY)Z \rangle &\Rightarrow P\langle XZ \rangle \langle YZ \rangle \\
(\text{zeta}) \langle XY \rangle &\Rightarrow (XY) \text{ if } \vdash X : A \rightarrow B, \vdash Y : A
\end{aligned}$$

**Facts:**

- (1) subject reduction
- (2) weak diamond
- (3) strong normalization
- (4) Church-Rosser

Definition of the faces  $f(X)$  of an untyped term  $X$ :

$$\begin{aligned}
f(x) &= \{x\} \\
f(\lambda x. X) &= \{\lambda x. Z \mid Z : f(X)\} \\
f(PXY) &= f(X) \cup f(Y) \\
f(LX) &= f(X) \\
f(RX) &= f(X) \\
f(XY) &= \{Z'Z'' \mid Z' : f(X) \text{ and } Z'' : f(Y)\}
\end{aligned}$$

Definition of a  $\sim$  regular term for a binary relation  $\sim$ :

$X$  is  $\sim$  regular if for any subterm  $PYZ$  of  $X$  and  $U : f(Y), V : f(Z)$  we have  $U \sim V$

Examples of  $\sim$

- (1) alpha conversion
- (2) eta conversion
- (3) beta-eta conversion

Stovring's notion of symmetric is the same as beta-eta regular. Here we note that the condition  $U : f(X), V : f(X) \Rightarrow U \sim V$  is not sufficient to ensure that  $X$  is  $\sim$  regular already for  $\sim = \text{eta}$ . For example,

$$\lambda y. Px((\lambda z. x)y)$$

has only one face  $\lambda z. x$ , modulo eta, but  $Px((\lambda z. x)y)$  has two. The faces of a term typed with simple types and Cartesian products are obtained by erasing the typing, changing  $\langle \rangle$  to  $( )$  and computing  $f$  of the result.

**Lemma 1.** *If, in BCD,  $A \leq B$  then there exist an eta regular  $x'$  such that in simple types with Cartesian products we have  $x : A' \vdash x' : B'$  and any face of  $x'$  eta reduces to  $x$ .*

*Proof.* By induction on the length of a BCD derivation of  $A \leq B$ .

Basis:

Case 1: (refl.) We set  $x' := x$

$$x : A \vdash x : A$$



Case 2: ( $\text{inc}_L$ ). We set  $x' := Lx$

$$\begin{aligned} x : A^*B &\vdash x : A^*B \\ x : A^*B &\vdash Lx : A \end{aligned}$$

Case 3: ( $\text{inc}_R$ ). We set  $x' := Rx$

$$\begin{aligned} x : A^*B &\vdash x : A^*B \\ x : A^*B &\vdash Rx : B \end{aligned}$$

Case 4: ( $\rightarrow \wedge$ ). We set  $x' := \lambda y. \langle xy \rangle$

$$\begin{aligned} x : (A \rightarrow B)^*(A \rightarrow C) &\vdash x : (A \rightarrow B)^*(A \rightarrow C) \\ x : (A \rightarrow B)^*(A \rightarrow C) &\vdash Lx : A \rightarrow B \\ x : (A \rightarrow B)^*(A \rightarrow C) &\vdash Rx : A \rightarrow C \\ y : A &\vdash y : A \end{aligned}$$

$$x : (A \rightarrow B)^*(A \rightarrow C), y : A \vdash \langle Lxy \rangle : B$$

$$x : (A \rightarrow B)^*(A \rightarrow C), y : A \vdash \langle Rxy \rangle : C$$

$$x : (A \rightarrow B)^*(A \rightarrow C), y : A \vdash \langle xy \rangle : B^*C$$

$$x : (A \rightarrow B)^*(A \rightarrow C) \vdash \lambda y. \langle xy \rangle : A \rightarrow (B^*C)$$

Induction step:

**Case 1:** ( $glb$ )

By induction hypothesis we may assume that we have  $x : C' \vdash x'1 : A'$  and  $x : C' \vdash x'2 : B'$  in simple types with surjective pairing. Thus, we can set  $x' := P(x'1)(x'2)$ .

**Case 2:** ( $\text{trans}$ )

By induction hypothesis we may assume that we have  $y : B' \vdash y'1 : C'$  and  $x : A' \vdash x'2 : B'$  in simple types with surjective pairing. Thus we can set  $[x'2/y](y'1)$

**Case 3:** ( $\rightarrow$ )

By induction hypothesis we may assume that we have  $y : C' \vdash y'1 : A'$  and  $z : B' \vdash z'2 : D$  in simple types with surjective pairing. Then we can set  $x' := \lambda y. (x(y'1))'2$  and we have

$$\begin{aligned}
x : A' \rightarrow B', y : C' &\vdash x(y'1) : B' \\
x(y'1) : B' &\vdash (x(y'1))'2 : D' \\
x : A' \rightarrow B', y : C' &\vdash (x(y'1))'2 : D' \\
x : A' \rightarrow B' &\vdash \lambda y. (x(y'1))'2 : C' \rightarrow D'
\end{aligned}$$

End of proof.  $\square$

**Theorem 1.** *If in BCD we have  $S \vdash X : A$  then there is eta regular  $X'$  such that in simple types with Cartesian products  $S' \vdash X' : A'$  and for any face  $U$  of  $X'$ ,  $U$  eta reduces to  $X$ .*

*Proof.* By induction on the length of a BCD derivation of  $S \vdash X : A$ .

Basis:  $(Ax)$  This case is trivial.

Induction step:

**Case 1:** The derivation ends in the  $[]$  rule. This is the content of Lemma 1.

**Case 2:** The derivation ends in  $\wedge E$ ,  $\rightarrow I$ , or  $\rightarrow E$ . This case follows directly the induction hypothesis.

**Case 3:** The derivation ends in  $\wedge I$ . So in BCD we have  $S \vdash X : A$  and  $S \vdash X : B$ . By induction hypothesis we have  $S' \vdash X'1 : A'$  and  $S' \vdash X'2 : B'$ . Thus  $S' \vdash P(x'1)(x'2) : (A \wedge B)'$  and  $P(x'1)(x'2)$  is eta regular. End of proof.  $\square$

**Theorem 2.** *If  $X$  is eta regular and in simple types with Cartesian products we have  $S \vdash X : A$  then for any face  $U$  of  $X$  there exists an eta reduct  $U''$  such that we have in BCD  $S'' \vdash U'' : A''$ .*

*Proof.* By induction on the length of a typing derivation of  $X$ .

Basis:  $X = x$ . Obvious.

**Case 1:**  $X = PYZ : A^*B$ .  $Y$  and  $Z$  are eta regular, and the induction hypothesis applies to them. Thus, for any  $U : f(Y)$  and  $V : f(Z)$  there exist eta reducts  $U''$  and  $V''$  respectively such that in BCD,  $S'' \vdash U'' : A''$  and  $S'' \vdash V'' : B''$ . Now  $U'' \sim V''$  so by subject reduction and Church-Rosser for eta there exists  $W''$  such that  $U''$  eta reduces to  $W''$  eta expands to  $V''$  and  $S'' \vdash W'' : A'' \wedge B''$  in BCD.

**Case 2:**  $X = LY$  or  $RY$ . By induction hypothesis and  $\wedge E$ .

**Case 3:**  $X = \lambda y. Y : A \rightarrow B$ . Now the induction hypothesis applies to  $S \cup \{y : A\} \vdash Y : B$  so  $S'' \cup \{y : A''\} \vdash Y'' : B''$ . Hence  $S'' \vdash \lambda y. Y'' : (A \rightarrow B)''$ .

**Case 4:**  $X = (YZ) : B$  with  $S \vdash Y : A \rightarrow B$ . As in Case 1 for any  $U : f(Y)$  and  $V : f(Z)$  there exist eta reducts  $U''$  and  $V''$  respectively such that in BCD,  $S'' \vdash U'' : A'' \rightarrow B''$  and  $S'' \vdash V'' : A''$ . Thus  $S'' \vdash U''V'' : B''$ .

**Case 5:**  $X = \langle YZ \rangle$ . As in Case 1 for any  $U : f(Y)$  and  $V : f(Z)$  there exist eta reducts  $U''$  and  $V''$  respectively such that in BCD,  $S'' \vdash U''V'' : A$ , and other eta reducts  $U'''$  and  $V'''$  respectively such that in BCD,  $S'' \vdash U'''V''' : B''$ . By Church-Rosser for eta and subject reductions there exists  $W$  such that  $UV$  eta reduces to  $U''V''$  eta reduces to  $W$  eta expands to  $U''V''$  eta expands to  $UV$  and  $S'' \vdash W : A \wedge B$ . End of proof.  $\square$

Now Theorems 1 and 2 combine for a nice characterization of the case when  $X$  is eta normal.

**Corollary 1.** *If  $X$  is eta normal, then in BCD we have  $S \vdash X : A$  if and only if there is an eta regular  $X'$  such that  $S' \vdash X' : A$  and  $X$  is an eta reduct of any face of  $X'$ .*

Remark: The eta reductions in Theorem 1 and 2 could be removed by the technique of [5]. There we extend the type structure to make  $\rightarrow$  “almost” surjective. For each atom  $p$  we add new atoms  $p_l, p_r$  and the definitional equality  $p := p_l \rightarrow p_r$ . Replacing an atom by its definition is referred to as “type expansion”. It is obvious that two distinct type expansions of a given type have a common type expansion. This was our original formulation of the result. However, here we prefer to state the outcome for the original BCD. With type expansions BPS becomes useful. We state without proof the useful lemmas.

**Lemma 2.** *If in BPS,  $S \vdash X : A$  then for any eta expansion  $X\#$  of  $X$  there exists type expansions  $S\#, A\#$  such that  $S\# \vdash X\# : A\#$ .*

**Lemma 3.** *If in BCD,  $S \vdash X : A$  then there is an eta expansion  $X\#$  of  $X$  and a type expansion  $S\#$  of  $S$  and  $A\#$  of  $A$  such that in BPS,*

$$S\# \vdash X\# : A\#$$

## References

- [1] Barendregt, H., Dekkers, ., Statman, R., *Lambda Calculus with Types*, CUP, (2013).
- [2] Bucciarelli, ., Piperno, ., Salvo ., Intersection types and lambda definability, *MSCS03*, **13** (1), pp. 15-53, (2003).
- [3] Coppo, ., Dezani, ., A new type assignment for lambda terms, *Archiv fur math. logik*, **19** (1), pp. 139-156, (1978).
- [4] de Vrijer, ., Extending the lambda calculus with surjective pairing is conservative, *LICS* **4**, pp. 204-215, (1989).
- [5] Statman, R., A local translation of untyped into simply typed lambda-calculus, CMU Research Report #91-134, (1991).
- [6] Statman, R., Surjective pairing revisited, in *Liber Armicorum for Roel DeVrijer*. Klop, van Oostrom, and van Raamsdonk, eds., University of Amsterdam, (2009).
- [7] Stovring, ., Extending the extensional lambda calculus with surjective pairing is conservative, *LMCS* **2**, pp. 1-14.

# Approximate Relational Hoare Logic for Continuous Random Samplings

Tetsuya Sato<sup>1</sup>

*Research Institute for Mathematical Sciences, Kyoto University, Kyoto, 606-8502, Japan*

---

## Abstract

Approximate relational Hoare logic (apRHL) is a logic for formal verification of the differential privacy of databases written in the programming language pWHILE. Strictly speaking, however, this logic deals only with discrete random samplings. In this paper, we define the graded relational lifting of the subprobabilistic variant of Giry monad, which described differential privacy. We extend the logic apRHL with this graded lifting to deal with continuous random samplings. We give a generic method to give proof rules of apRHL for continuous random samplings.

*Keywords:* Differential privacy, Giry monad, graded monad, relational lifting, semantics,

---

## 1 Introduction

Differential privacy is a *definition* of privacy of *randomized* databases proposed by Dwork, McSherry, Nissim and Smith [7]. A randomized database satisfies  $\varepsilon$ -differential privacy (written  $\varepsilon$ -differentially private) if for any two adjacent data, the difference of their output probability distributions is bounded by the privacy strength  $\varepsilon$ . Differential privacy guarantees high secrecy against database attacks regardless of the attackers' background knowledge, and it has the composition laws, with which we can calculate the privacy strength of a composite database from the privacy strengths of its components.

*Approximate relational Hoare logic* (apRHL) [2,16] is a probabilistic variant of the *relational Hoare logic* [4] for formal verification of the differential privacy of databases written in the programming language pWHILE. In the logic apRHL, a parametric relational lifting, which relate probability distributions, play a central role to describe differential privacy in the framework of verification. This parametric lifting is an extension of the relational lifting [10, Section 3] that captures probabilistic bisimilarity of Markov chains [13] (see also [6, lemma 4]). The concept

---

<sup>1</sup> Email: [satoutet@kurims.kyoto-u.ac.jp](mailto:satoutet@kurims.kyoto-u.ac.jp)

of differential privacy is described in the category of binary relation and mappings between them, and verified by the logic apRHL.

Strictly speaking, however, apRHL deals only with random samplings of *discrete* distributions, while the algorithms in many actual studies for differential privacy are modelled with *continuous* distributions, such as, the Laplacian distributions over real line. Therefore apRHL is desired to be extended to deal with random continuous samplings.

### 1.1 Contributions

Main contributions of this paper are the following two points:

- We define the graded relational lifting of sub-Giry monad describing differential privacy for continuous random samplings.
- We extend the logic apRHL [2,16] for continuous random samplings (we name *continuous apRHL*).

This graded relational lifting is developed without witness distributions of probabilistic coupling, and hence is constructed in a different way from the coupling-based parametric lifting of relations given in the studies of apRHL [1,2,16].

In the continuous apRHL, we mainly extend the proof rules for relation compositions and the frame rule. We also develop a generic method to construct proof rules for random samplings. By importing the new rules added to apRHL+ in [1], we give a formal proof of the differential privacy of the *above-threshold algorithm* for real-valued queries [8, Section 3.6].

### 1.2 Preliminaries

We denote by **Meas** the category of measurable spaces and measurable functions between them and denote by **Set** the category of all sets and functions. The category **Meas** is complete and cocomplete, and the forgetful functor  $U: \mathbf{Meas} \rightarrow \mathbf{Set}$  preserves products and coproducts. We also denote by  $\omega\mathbf{CPO}_\perp$  of the category of  $\omega$ -complete partial orders with the least element and continuous functions.

## A Category of Relations between Measurable Spaces

We introduce the category **BRel**(**Meas**) of binary relations between measurable spaces as follows:

- An object is a triple  $(X, Y, \Phi)$  consisting of measurable spaces  $X$  and  $Y$  and a relation  $\Phi$  between  $X$  and  $Y$  (i.e.  $\Phi \subseteq UX \times UY$ ). We remark that  $\Phi$  does not need to be a measurable subset of the product space  $X \times Y$ .
- An arrow  $(f, g): (X, Y, \Phi) \rightarrow (X', Y', \Phi')$  is a pair of measurable functions  $f: X \rightarrow X'$  and  $g: Y \rightarrow Y'$  such that  $(Uf \times Ug)(\Phi) \subseteq \Phi'$ .

When we write an object  $(X, Y, \Phi)$  in **BRel**(**Meas**), we omit writing the underlying spaces  $X$  and  $Y$  if they are obvious from the context. We write  $p$  for the forgetful functor  $p: \mathbf{BRel}(\mathbf{Meas}) \rightarrow \mathbf{Meas} \times \mathbf{Meas}$  which extracting underlying spaces:  $(X, Y, \Phi) \mapsto (X, Y)$ . We call an endofunctor  $F$  on **BRel**(**Meas**) a *relational lifting* of an endofunctor  $E$  on **Meas** if  $(E \times E)p = pF$ .

### The Sub-Giry Monad

The Giry monad on **Meas** is introduced in [9] to give a categorical approach to probability theory; each arrow  $X \rightarrow Y$  in the Kleisli category of the Giry monad bijectively corresponds to a probabilistic transition from  $X$  to  $Y$ , and the Chapman-Kolmogorov equation corresponds to the associativity law of the Giry monad.

We recall the sub-probabilistic variant of the Giry monad, which we call the *sub-Giry monad* (see also [17, Section 4]):

- For any measurable space  $(X, \Sigma_X)$ , the measurable space  $(\mathcal{G}X, \Sigma_{\mathcal{G}X})$  is defined as follows: the underlying set  $\mathcal{G}X$  is the set of subprobability measures over  $X$ , and the  $\sigma$ -algebra  $\Sigma_{\mathcal{G}X}$  is the coarsest one that makes the evaluation function  $\text{ev}_A: \mathcal{G}X \rightarrow [0, 1]$  (mapping  $\nu$  to  $\nu(A)$ ) measurable for each  $A \in \Sigma_X$ .
- For each  $f: X \rightarrow Y$  in **Meas**,  $\mathcal{G}f: \mathcal{G}X \rightarrow \mathcal{G}Y$  is defined by  $(\mathcal{G}f)(\nu) = \nu(f^{-1}(-))$ .
- The unit  $\eta$  is defined by  $\eta_X(x) = \delta_x$ , where  $\delta_x$  is the *Dirac measure* centred on  $x$ .
- The multiplication  $\mu$  is defined by  $\mu_X(\Xi)(A) = \int_{\mathcal{G}X} \text{ev}_A d(\Xi)$ . The Kleisli lifting of  $f: X \rightarrow \mathcal{G}Y$  is given by  $f^\sharp(\nu)(A) = \int_X f(-)(A) d\nu$  ( $\nu \in \mathcal{G}X$ ).

The monad  $\mathcal{G}$  is commutative strong with respect to the cartesian product in **Meas**. The strength  $\text{st}_{-,=} : (-) \times \mathcal{G}(=) \Rightarrow \mathcal{G}(- \times =)$  is given by the product measure  $\text{st}_{X,Y}(x, \nu) = \delta_x \otimes \nu$ . The commutativity of  $\mathcal{G}$  is given from the Fubini theorem. The double strength  $\text{dst}_{-,=} : \mathcal{G}(-) \times \mathcal{G}(=) \Rightarrow \mathcal{G}(- \times =)$  is given by  $\text{dst}_{X,Y}(\nu_1, \nu_2) = \nu_1 \otimes \nu_2$ .

The Kleisli category **Meas $_{\mathcal{G}}$**  is often called the category **SRel** of *stochastic relations* [17, Section 3]. The category **SRel** is  $\omega\mathbf{CPO}_{\perp}$ -enriched (with respect to the cartesian monoidal structure) with the following pointwise order:

$$f \sqsubseteq g \iff \forall x \in X, B \in \Sigma_Y. f(x)(B) \leq g(x)(B) \quad (f, g: X \rightarrow Y \text{ in } \mathbf{SRel}).$$

The *least upper bound*  $\sup_{n \in \mathbb{N}} f_n$  of any  $\omega$ -chain  $f_0 \sqsubseteq f_1 \sqsubseteq \dots \sqsubseteq f_n \sqsubseteq \dots$  is given by  $(\sup_n f_n)(x)(B) = \sup_n (f_n(x)(B))$ . The *least function* of each **SRel**( $X, Y$ ) (written  $\perp_{X,Y}$ ) is the constant function of the null-measure over  $Y$ .

This enrichment is equivalent to the partially additive structure on **SRel** [17, Section 5]: For any  $\omega$ -chain  $\{f_n\}_{n \in \mathbb{N}}$  of  $f_n: X \rightarrow Y$  in **SRel**, we have the summable sequence  $\{g_n\}_n$  where  $g_0 = f_0$  and  $g_{n+1} = f_{n+1} - f_n$ . Conversely, for any summable sequence  $\{g_n\}_{n \in \mathbb{N}}$ , the functions  $f_n = \sum_{k=0}^n g_k$  form an  $\omega$ -chain.

### Differential privacy

Throughout this paper, we define the approximate differential privacy as follows:

**Definition 1.1** [[8, Definition 2.4], Modified] A measurable function  $c: \mathbb{R}^m \rightarrow \mathcal{G}(\mathbb{R}^n)$  is  $(\varepsilon, \delta)$ -differentially private if  $c(x)(A) \leq \exp(\varepsilon)c(y)(A) + \delta$  holds for any  $\|x - y\|_1 \leq 1$  and  $A \in \Sigma_{\mathbb{R}^n}$ , where  $\|\cdot\|_1$  is 1-norm of the Euclidean space  $\mathbb{R}^m$ .

What we modify from the original definition [8, Definition 2.4] is the domain and codomain of  $c$ ; we replace the domain from  $\mathbb{N}$  to  $\mathbb{R}$ , and replace the codomain from a discrete probability space to  $\mathcal{G}(\mathbb{R}^n)$ . We apply this definition to the interpretation of pWHILE programs. The input and output spaces can be other spaces: in section 4 we consider the *above-threshold algorithm* **Above** whose output space is  $\mathbb{Z}$ . The

above modification is essential in describing and verifying the differential privacy of this algorithm because it takes a sample from Laplace distribution over *real line*.

## 2 A Graded Monad for Differential Privacy

The composition law of differential privacy plays crucial role to in the compositional verification of the differential privacy of database programs. Barthe, Köpf, Olmedo, and Zanella-Béguelin constructed a *parametric relational lifting* describing differential privacy, and developed a framework for compositional verification of differential privacy [2].

Following this relational approach, we construct the parametric relational lifting of Giry monad to describe differential privacy for continuous random samplings. This lifting forms a graded monad on the category  $\mathbf{BRel}(\mathbf{Meas})$  in the sense of [11]. The axioms of graded monad correspond to the (sequential) composition law of differential privacy.

### 2.1 Graded Monads

**Definition 2.1** [11, Definition 2.2-bis] Let  $\mathbb{C}$  be a category, and  $(M, \cdot, 1, \preceq)$  be a *preordered monoid*. An  $M$ -graded (or  $M$ -parametric effect) monad on  $\mathbb{C}$  consists of

- a collection  $\{T_e\}_{e \in M}$  of endofunctors on  $\mathbb{C}$ ,
- a natural transformation  $\eta: \text{Id} \Rightarrow T_1$ ,
- a collection  $\{\mu^{e_1, e_2}\}_{e_1, e_2 \in M}$  of natural transformations  $\mu^{e_1, e_2}: T_{e_1}T_{e_2} \Rightarrow T_{e_1 e_2}$ ,
- a collection  $\{\sqsubseteq^{e_1, e_2}\}_{e_1 \preceq e_2}$  of natural transformations  $\sqsubseteq^{e_1, e_2}: T_{e_1} \Rightarrow T_{e_2}$

satisfying

- $\mu^{e, 1} \circ T_e \eta = \mu^{1, e} \circ \eta_{T_e} = \text{Id}_{T_e}$  for any  $e \in M$ ,
- $\mu^{(e_1 e_2), e_3} \circ \mu^{e_1, e_2} T_{e_3} = \mu^{e_1, (e_2 e_3)} \circ T_{e_1} \mu^{e_2, e_3}$  for all  $e_1, e_2, e_3 \in M$ ,
- $\sqsubseteq^{e, e} = \text{Id}_{T_e}$  for any  $e$  and  $\sqsubseteq^{e_2, e_3} \circ \sqsubseteq^{e_1, e_2} = \sqsubseteq^{e_1, e_3}$  whenever  $e_1 \preceq e_2 \preceq e_3$ ,
- $\sqsubseteq^{(e_1 e_2), (e_3 e_4)} \circ \mu^{e_1, e_2} = \mu^{e_3, e_4} \circ (\sqsubseteq^{e_1, e_3} * \sqsubseteq^{e_2, e_4})$  whenever  $e_1 \preceq e_3$  and  $e_2 \preceq e_4$ .

An  $M$ -graded monad  $(\{T_e\}_{e \in M}, \eta, \mu^{e_1, e_2}, \sqsubseteq^{e_1, e_2})$  on  $\mathbb{C}$  is called an  $M$ -graded lifting of monad  $(T, \eta^T, \mu^T)$  on  $\mathbb{D}$  along  $U: \mathbb{C} \rightarrow \mathbb{D}$  if  $UT_e = TU$ ,  $U(\eta) = \eta^T U$ ,  $U(\mu^{e_1, e_2}) = \mu^T U$ , and  $U(\sqsubseteq^{e_1, e_2}) = \text{id}_T$ .

### 2.2 A Graded Relational Lifting of Giry Monad for Differential Privacy

Let  $M$  be the cartesian product of the monoids  $([1, \infty), \times, 1)$  and  $([0, \infty), +, 0)$  equipped with the product order of numerical orders. For each  $(\gamma, \delta) \in M$ , we define the following mapping of  $\mathbf{BRel}(\mathbf{Meas})$ -objects by

$$\mathcal{G}^{(\gamma, \delta)} \Phi = \left\{ (\nu_1, \nu_2) \in \mathcal{G}X \times \mathcal{G}Y \left| \begin{array}{l} \forall A \in \Sigma_X, B \in \Sigma_Y. \\ \Phi(A) \subseteq B \implies \nu_1(A) \leq \gamma \nu_2(B) + \delta \end{array} \right. \right\}.$$



**Proposition 2.2**  $\{\mathcal{G}^{(\gamma,\delta)}\}_{(\gamma,\delta) \in M}$  forms an  $M$ -graded lifting of the monad  $(\mathcal{G} \times \mathcal{G}, \eta \times \eta, \mu \times \mu)$  along the forgetful functor  $p: \mathbf{BRel}(\mathbf{Meas}) \rightarrow \mathbf{Meas} \times \mathbf{Meas}$ .

**Proof.** Since the functor  $p$  is faithful, it suffices to show:

- (i) Each  $\mathcal{G}^{(\gamma,\delta)}$  is an endofunctor on  $\mathbf{BRel}(\mathbf{Meas})$ .
  - (ii)  $(\text{id}_{\mathcal{G}X}, \text{id}_{\mathcal{G}Y})$  is an arrow  $\mathcal{G}^{(\gamma,\delta)}\Phi \rightarrow \mathcal{G}^{(\gamma',\delta')}\Phi$  in  $\mathbf{BRel}(\mathbf{Meas})$  for all  $\gamma, \gamma', \delta, \delta'$  such that  $\gamma \leq \gamma'$  and  $\delta \leq \delta'$ .
  - (iii)  $(\eta_X, \eta_Y)$  is an arrow  $\Phi \rightarrow \mathcal{G}^{(1,0)}\Phi$  in  $\mathbf{BRel}(\mathbf{Meas})$ .
  - (iv)  $(\mu_X, \mu_Y)$  is an arrow  $\mathcal{G}^{(\gamma,\delta)}\mathcal{G}^{(\gamma',\delta')}\Phi \rightarrow \mathcal{G}^{(\gamma\gamma',\delta+\delta')}\Phi$  in  $\mathbf{BRel}(\mathbf{Meas})$  for all  $\gamma, \gamma', \delta, \delta'$ .
- (i) Since the mapping  $(f, g) \mapsto (\mathcal{G}f, \mathcal{G}g)$  is obviously functorial, it suffices to check that  $(\mathcal{G}f, \mathcal{G}g)$  is an arrow  $\mathcal{G}^{(\gamma,\delta)}\Psi \rightarrow \mathcal{G}^{(\gamma,\delta)}\Phi$  in  $\mathbf{BRel}(\mathbf{Meas})$  for any arrow  $(f, g): \Psi \rightarrow \Phi$  in  $\mathbf{BRel}(\mathbf{Meas})$ . This is proved from  $\Phi(A) \subseteq B \implies \Psi(f^{-1}(A)) \subseteq g^{-1}(B)$  for any  $A \in \Sigma_X$  and  $B \in \Sigma_Y$ . (ii) Obvious. (iii) Obvious. (iv) It suffices to show  $(\mu_X \times \mu_Y)(\mathcal{G}^{(\gamma,\delta)}\mathcal{G}^{(\gamma',\delta')}\Phi) \subseteq \mathcal{G}^{(\gamma\gamma',\delta+\delta')}\Phi$  for any  $\Phi \subseteq X \times Y$ .

First, the following equation holds:

$$\mathcal{G}^{(\gamma,\delta)}\Phi = \left\{ (\nu_1, \nu_2) \mid \forall (f, g): \Phi \rightarrow \leq \text{ in } \mathbf{BRel}(\mathbf{Meas}). \int_X f d\nu_1 \leq \gamma \int_Y g d\nu_2 + \delta \right\},$$

where  $\leq$  is the numerical order relation on  $\mathcal{G}1 \simeq [0, 1]$ . We omit the proof of this equation. It can be shown in the same way as [12, Theorem 12].

Let  $(\Xi_1, \Xi_2) \in \mathcal{G}^{(\gamma,\delta)}\mathcal{G}^{(\gamma',\delta')}\Phi$ . Assume  $\Phi(A) \subseteq B$ . We give  $(f, g): \mathcal{G}^{(\gamma',\delta')}\Phi \rightarrow \leq$  in  $\mathbf{BRel}(\mathbf{Meas})$  by  $f = \max(\text{ev}_A - \delta', 0)$  and  $g = \min(\gamma' \cdot \text{ev}_B, 1)$ . They actually satisfy  $f(\nu_1) \leq g(\nu_2)$  for each  $(\nu_1, \nu_2) \in \mathcal{G}^{(\gamma',\delta')}\Phi$ . Hence,

$$\begin{aligned} \mu_X(\Xi_1)(A) - \delta' &\leq \int_{\mathcal{G}X} (\text{ev}_A - \delta') d\Xi_1 \leq \int_{\mathcal{G}X} f d\Xi_1 \\ &\leq \gamma \int_{\mathcal{G}X} g d\Xi_2 + \delta \leq \gamma \int_{\mathcal{G}X} \gamma' \text{ev}_B d\Xi_2 + \delta = \gamma\gamma' \mu_Y(\Xi_2)(B) + \delta. \end{aligned}$$

This implies  $\mu_X(\Xi_1)(A) \leq \gamma\gamma' \mu_Y(\Xi_2)(B) + \delta + \delta'$ .  $\square$

**Proposition 2.3 (Composability laws for differential privacy)**

- For any  $(f_1, g_1): \Phi_1 \rightarrow \mathcal{G}^{(\gamma,\delta)}\Psi_1$  and  $(f_2, g_2): \Phi_2 \rightarrow \mathcal{G}^{(\gamma',\delta')}\Psi_2$  in  $\mathbf{BRel}(\mathbf{Meas})$ ,  $(\text{dst} \circ (f_1 \times f_2), \text{dst} \circ (g_1 \times g_2))$  is an arrow  $\Phi_1 \dot{\times} \Phi_2 \rightarrow \mathcal{G}^{(\gamma\gamma',\delta+\delta')}(\Psi_1 \dot{\times} \Psi_2)$  in  $\mathbf{BRel}(\mathbf{Meas})$ .
- For any  $(f_1, g_1): \Phi_1 \rightarrow \mathcal{G}^{(\gamma,\delta)}\Psi$  and  $(f_2, g_2): \Phi_2 \rightarrow \mathcal{G}^{(\gamma',\delta')}\Psi$  in  $\mathbf{BRel}(\mathbf{Meas})$ ,  $([f_1, f_2], [g_1, g_2])$  is an arrow  $\Phi_1 \dot{+} \Phi_2 \rightarrow \mathcal{G}^{(\max(\gamma,\gamma'), \max(\delta,\delta'))}\Psi$  in  $\mathbf{BRel}(\mathbf{Meas})$ .

Here,  $\dot{\times}$  and  $\dot{+}$  are product and coproduct in  $\mathbf{BRel}(\mathbf{Meas})$  respectively.

**Theorem 2.4** A measurable function  $c: \mathbb{R}^m \rightarrow \mathcal{G}(\mathbb{R}^n)$  is  $(\varepsilon, \delta)$ -differentially private if and only if  $(c, c)$  is an arrow  $\{(x, y) \mid \|x - y\|_1 \leq 1\} \rightarrow \mathcal{G}^{(\exp(\varepsilon), \delta)}\text{Eq}_{\mathbb{R}^n}$  in  $\mathbf{BRel}(\mathbf{Meas})$ .

We remark that the relations  $\{(x, y) \mid \|x - y\|_1 \leq 1\}$  and  $\text{Eq}_{\mathbb{R}^n}$  are symmetric. Thus, the lifting  $\{\mathcal{G}^{(\gamma,\delta)}\}_{(\gamma,\delta) \in M}$  describes only one side of inequalities in the defi-

inition of differential privacy. By symmetrising this lifting, We obtain an  $M$ -graded lifting  $\{\mathcal{G}^{(\gamma,\delta)}\}_{(\gamma,\delta) \in M}$  exactly describing the differential privacy for continuous probabilities:

$$\overline{\mathcal{G}^{(\gamma,\delta)}} = \mathcal{G}^{(\gamma,\delta)}(-) \cap (\mathcal{G}^{(\gamma,\delta)}(-)^{\mathfrak{P}})^{\mathfrak{P}}.$$

In the original works [2,3] of apRHL, the following relational lifting  $(-)^{\sharp(\gamma,\delta)}$  is introduced to describe differential privacy. This lifting relates two distributions if there are intermediate distributions  $d_L$  and  $d_R$ , called *witnesses*, whose skew distance, defined by  $\Delta_\gamma^X(d_L, d_R) = \sup_{C \subseteq X} \max(d_L(C) - \gamma d_R(C), d_R(C) - \gamma d_L(C), 0)$ , is less than or equal to  $\delta$ .

**Definition 2.5** ([3, Definition 4], [16, Definition 4.3] and [1, Definition 8]) We denote by  $\mathcal{D}$  the subdistribution monad over **Set**. Let  $\Psi$  be a relation between sets  $X$  and  $Y$ , and  $d_L \in \mathcal{D}X$  and  $d_R \in \mathcal{D}Y$  be two subdistributions. We define the relation  $\Psi^{\sharp(\gamma,\delta)} \subseteq \mathcal{D}X \times \mathcal{D}Y$  as follows:  $(d_L, d_R) \in \Psi^{\sharp(\gamma,\delta)}$  if and only if there are two subdistributions  $d_L, d_R \in \mathcal{D}(X \times Y)$ , called *witnesses*, such that

$$\mathcal{D}\pi_1(d_L) = d_L, \mathcal{D}\pi_2(d_R) = d_R, \text{supp}(d_L) \subseteq \Psi, \text{supp}(d_R) \subseteq \Psi, \Delta_\gamma^{X \times Y}(d_L, d_R) \leq \delta.$$

**Proposition 2.6** For any countable discrete spaces  $X$  and  $Y$ , and relation  $\Psi \subseteq X \times Y$ , we have  $\Psi^{\sharp(\gamma,\delta)} \subseteq \overline{\mathcal{G}^{(\gamma,\delta)}}\Psi$ .

**Proof.** Suppose  $(d_L, d_R) \in \Psi^{\sharp(\gamma,\delta)}$  with witnesses  $d_L$  and  $d_R$ . For any  $A \subseteq X$ , since  $\text{supp}(d_L) \subseteq \Psi$  and  $(A \times Y) \cap \Psi \subseteq X \times \Psi(A)$ , we obtain:

$$\begin{aligned} d_L(A) &= \mathcal{D}\pi_1(d_L)(A) = d_L(A \times Y) = d_L((A \times Y) \cap \Psi) \leq d_L(X \times \Psi(A)) \\ &\leq \gamma d_R(X \times \Psi(A)) + \delta = \gamma \mathcal{D}\pi_2(d_R)(\Psi(A)) + \delta = \gamma d_R(\Psi(A)) + \delta. \end{aligned}$$

This implies  $(d_L, d_R) \in \overline{\mathcal{G}^{(\gamma,\delta)}}\Psi$ . Since the construction of  $(-)^{\sharp(\gamma,\delta)}$  is symmetric, we conclude  $(d_L, d_R) \in \overline{\mathcal{G}^{(\gamma,\delta)}}\Psi$ .  $\square$

We remark  $\mathcal{G}X = \mathcal{D}X$  for countable discrete space  $X$ . When  $X$  is not countable, we have the above results by embedding each  $d \in \mathcal{D}X$  in the set  $\mathcal{D}X'$  of subprobability distributions over the countable subspace  $X' = X \cap \text{supp}(d)$ .

**Corollary 2.7** We have  $\text{Eq}_X^{\sharp(\gamma,\delta)} = \overline{\mathcal{G}^{(\gamma,\delta)}}\text{Eq}_X$  for each countable discrete space  $X$ .

**Proof.** ( $\subseteq$ ) This inclusion is given from Proposition 2.6. ( $\supseteq$ ) Suppose  $(d_L, d_R) \in \overline{\mathcal{G}^{(\gamma,\delta)}}\text{Eq}_X$ . This is equivalent to  $\Delta_\gamma^X(d_L, d_R) \leq \delta$ . Hence  $(d_L, d_R) \in \text{Eq}_X^{\sharp(\gamma,\delta)}$  is proved by the witnesses given by  $d_L = \sum_{x \in X} d_L(x) \cdot \delta_{(x,x)}$  and  $d_R = \sum_{x \in X} d_R(x) \cdot \delta_{(x,x)}$ .  $\square$

### 3 The Continuous apRHL

We introduce a variant of the approximate probabilistic relational Hoare logic (apRHL) to deal with continuous random samplings. We name it the *continuous apRHL*.

#### 3.1 The Language $p\text{WHILE}$

We recall and reformulate categorically the language  $p\text{WHILE}$  [2]. In this paper, we mainly refer to the categorical semantics of a probabilistic language given in [5,

Section 2]. The language pWHILE is constructed in the standard way, hence we sometimes omit the details of its construction.

### 3.1.1 Syntax

We introduce the syntax of pWHILE by the following BNF:

$$\begin{aligned}
 \tau &::= \text{bool} \mid \text{int} \mid \text{real} \mid \dots \\
 e &::= x \mid p(e_1, \dots, e_m) \\
 \nu &::= d(e_1, \dots, e_m) \\
 i &::= x \leftarrow e \mid x \stackrel{\$}{\leftarrow} \nu \mid \text{if } e \text{ then } c_1 \text{ else } c_2 \mid \text{while } e \text{ do } c \\
 c &::= \text{skip} \mid \text{null} \mid \mathcal{I}; \mathcal{C}
 \end{aligned}$$

Here,  $\tau$  is a *value type*;  $x$  is a *variable*;  $p$  is an *operation*;  $d$  is a *probabilistic operation*;  $e$  is an *expression*;  $\nu$  is a *probabilistic expression*;  $i$  is an *imperative*;  $c$  is a *command* (or program). We remark constants are 0-ary operations.

We introduce the following syntax sugars for simplicity:

$$\begin{aligned}
 \text{if } b \text{ then } c &= \text{if } b \text{ then } c \text{ else skip} \\
 [\text{while } b \text{ do } c]_n &= \begin{cases} \text{if } b \text{ then null else skip,} & \text{if } n = 0 \\ \text{if } b \text{ then } c; [\text{while } b \text{ do } c]_k, & \text{if } n = k + 1 \end{cases}
 \end{aligned}$$

### 3.1.2 Typing Rules

We introduce a typing rule on the language pWHILE. A typing context is a finite set  $\Gamma = \{x_1: \tau_1, x_2: \tau_2, \dots, x_n: \tau_n\}$  of pairs of a variable and a value type such that each variable occurs only once in the context.

We give typing rules of pWHILE as follows:

$$\begin{aligned}
 &\frac{\Gamma \vdash^t e_1: \tau_1 \dots \Gamma \vdash^t e_n: \tau_n \quad p: (\tau_1, \dots, \tau_n) \rightarrow \tau}{\Gamma \vdash^t p(e_1, \dots, e_n): \tau} \quad \frac{\Gamma, x: \tau \vdash^t e: \tau}{\Gamma, x: \tau \vdash x \leftarrow e} \quad \frac{}{\Gamma \vdash \text{skip}} \\
 &\frac{x: \tau \in \Gamma \quad \Gamma \vdash^t e_1: \tau_1 \dots \Gamma \vdash^t e_n: \tau_n \quad d: (\tau_1, \dots, \tau_n) \rightarrow \tau}{\Gamma \vdash x \stackrel{\$}{\leftarrow} d(e_1, \dots, e_n): \tau} \quad \frac{}{\Gamma \vdash \text{null}} \\
 &\frac{\Gamma \vdash i \quad \Gamma \vdash c}{\Gamma \vdash i; c} \quad \frac{\Gamma \vdash^t b: \text{bool} \quad \Gamma \vdash c_1 \quad \Gamma \vdash c_2}{\Gamma \vdash \text{if } b \text{ then } c_1 \text{ else } c_2} \quad \frac{\Gamma \vdash^t b: \text{bool} \quad \Gamma \vdash c}{\Gamma \vdash \text{while } b \text{ do } c}
 \end{aligned}$$

Here, the type  $(\tau_1, \dots, \tau_n) \rightarrow \tau$  of each operation  $p$  and each probabilistic operation  $d$  are assumed to be given in advance.

We easily define inductively the set of free variables of commands, expressions, and probabilistic expressions (denoted by  $FV(c)$ ,  $FV(e)$ , and  $FV(\nu)$ ).

### 3.1.3 Denotational Semantics

We introduce a denotational semantics of pWHILE in **Meas**. We give the interpretations  $\llbracket \tau \rrbracket$  of the value types  $\tau$ :

- $\llbracket \text{bool} \rrbracket = \mathbb{B} = 1 + 1 = \{\text{true}, \text{false}\}$  (discrete space)
- $\llbracket \text{int} \rrbracket = \mathbb{Z}$  (discrete space)

- $\llbracket \text{real} \rrbracket = \mathbb{R}$  (Lebesgue measurable space)

We interpret a typing context  $\Gamma = \{x_1 : \tau_1, x_2 : \tau_2, \dots, x_n : \tau_n\}$  as the product space  $\llbracket \tau_1 \rrbracket \times \llbracket \tau_2 \rrbracket \times \dots \times \llbracket \tau_n \rrbracket$ . We interpret each operation  $p : (\tau_1, \dots, \tau_m) \rightarrow \tau$  as a measurable function  $\llbracket p \rrbracket : \llbracket \tau_1 \rrbracket \times \dots \times \llbracket \tau_m \rrbracket \rightarrow \llbracket \tau \rrbracket$ , and each probabilistic operation  $d : (\tau_1, \dots, \tau_m) \rightarrow \tau$  as  $\llbracket d \rrbracket : \llbracket \tau_1 \rrbracket \times \dots \times \llbracket \tau_m \rrbracket \rightarrow \mathcal{G}[\tau]$ . Typed terms  $\Gamma \vdash^t e : \tau$  and commands  $\Gamma \vdash c$  are interpreted to measurable functions of the forms  $\llbracket \Gamma \rrbracket \rightarrow \llbracket \tau \rrbracket$  and  $\llbracket \Gamma \rrbracket \rightarrow \mathcal{G}[\Gamma]$  respectively.

The interpretation of expressions are defined inductively by:

$$\llbracket \Gamma \vdash^t x : \tau \rrbracket = \pi_{x : \tau} \quad \llbracket \Gamma \vdash^t p(e_1, \dots, e_m) \rrbracket = \llbracket p \rrbracket(\llbracket \Gamma \vdash^t e_1 \rrbracket, \dots, \llbracket \Gamma \vdash^t e_m \rrbracket)$$

The interpretation of commands are defined inductively by:

$$\begin{aligned} \llbracket \Gamma \vdash \text{skip} \rrbracket &= \eta_{\llbracket \Gamma \rrbracket} \quad \llbracket \Gamma \vdash \text{null} \rrbracket = \perp_{\llbracket \Gamma \rrbracket, \llbracket \Gamma \rrbracket} \quad \llbracket \Gamma \vdash i; c \rrbracket = (\llbracket \Gamma \vdash c \rrbracket)^\sharp \circ \llbracket \Gamma \vdash i \rrbracket \\ \llbracket \Gamma \vdash x \stackrel{\$}{\leftarrow} d(e_1, \dots, e_m) \rrbracket &= \mathcal{G}(\rho_{(x : \tau, \Gamma)}) \circ \text{st}_{\llbracket \tau \rrbracket, \llbracket \Gamma \rrbracket} \circ \langle \llbracket d \rrbracket(\llbracket \Gamma \vdash^t e_1 \rrbracket, \dots, \llbracket \Gamma \vdash^t e_m \rrbracket), \text{id}_{\llbracket \Gamma \rrbracket} \rangle \\ \llbracket \Gamma, x : \tau \vdash x \leftarrow e \rrbracket &= \eta_{\llbracket \Gamma, x : \tau \rrbracket} \circ \rho_{(x : \tau, \Gamma)} \circ \langle \llbracket \Gamma, x : \tau \vdash e \rrbracket, \text{id}_{\llbracket \Gamma, x : \tau \rrbracket} \rangle \\ \llbracket \Gamma \vdash \text{if } b \text{ then } c_1 \text{ else } c_2 \rrbracket &= [\llbracket \Gamma \vdash c_1 \rrbracket, \llbracket \Gamma \vdash c_2 \rrbracket] \circ \cong_{\llbracket \Gamma \rrbracket} \circ \langle \llbracket \Gamma \vdash b \rrbracket, \text{id}_{\llbracket \Gamma \rrbracket} \rangle \\ \llbracket \Gamma \vdash \text{while } b \text{ do } c \rrbracket &= \sup_{n \in \mathbb{N}} \llbracket \Gamma \vdash [\text{while } e \text{ do } c]_n \rrbracket \end{aligned}$$

Here,

- $\rho_{(x_k : \tau_k, \Gamma)} = \langle f_l \rangle_{l \in \{1, 2, \dots, n\}} : \llbracket \tau_k \rrbracket \times \llbracket \Gamma \rrbracket \rightarrow \llbracket \Gamma \rrbracket$ , where  $\Gamma = \{x_1 : \tau_1, x_2 : \tau_2, \dots, x_n : \tau_n\}$ ,  $f_k = \pi_2$ , and  $f_l = \pi_l \circ \pi_2$  ( $l \neq k$ ).
- $\cong_X : 2 \times X \rightarrow X + X$  is the inverse of  $[\langle \iota_1 \circ !_X, \text{id} \rangle, \langle \iota_2 \circ !_X, \text{id} \rangle] : X + X \rightarrow 2 \times X$ , which is obtained from the distributivity of the category **Meas**.

We remark that, from the commutativity of the monad  $\mathcal{G}$ , if  $\Gamma \vdash x : \tau$  and  $x \notin FV(c)$  then  $\llbracket \Gamma \vdash c \rrbracket \cong \text{dst}_{\llbracket \Gamma' \rrbracket, \llbracket \tau \rrbracket}(\llbracket \Gamma' \vdash c \rrbracket \times \eta_{\llbracket \tau \rrbracket})$  where  $\Gamma' = \Gamma \setminus \{x : \tau\}$ .

### 3.2 Judgements of apRHL

A judgement of apRHL is

$$c_1 \sim_{\gamma, \delta} c_2 : \Psi \Rightarrow \Phi,$$

where  $c_1$  and  $c_2$  are commands, and  $\Psi$  and  $\Phi$  are objects in **BRel(Meas)**. We call the relations  $\Psi$  and  $\Phi$  the *precondition* and *postcondition* of the judgement respectively. Inspired from the validity of asymmetric apRHL [2], we introduce the validity of the judgement of apRHL.

**Definition 3.1** Let  $\Psi$  and  $\Phi$  be relations over the space  $\llbracket \Gamma \rrbracket$ . A judgement  $c_1 \sim_{\gamma, \delta} c_2 : \Psi \Rightarrow \Phi$  is valid (written  $\models c_1 \sim_{\gamma, \delta} c_2 : \Psi \Rightarrow \Phi$ ) when  $(\llbracket \Gamma \vdash c_1 \rrbracket, \llbracket \Gamma \vdash c_2 \rrbracket)$  is an arrow  $\Psi \rightarrow \overline{\mathcal{G}^{(\gamma, \delta)}}\Phi$  in **BRel(Meas)**.

We often write preconditions and postconditions in the following manner: Let  $\Gamma = \{x_1 : \tau_1, x_2 : \tau_2, \dots, x_n : \tau_n\}$ . Assume  $\Gamma \vdash e_1 : \tau$  and  $\Gamma \vdash e_2 : \tau$ , and let  $R$  be a relation on  $\llbracket \tau \rrbracket$  (e.g.  $=, \leq, \dots$ ). We define the relation  $e_1 \langle 1 \rangle R e_2 \langle 2 \rangle$  on  $\llbracket \Gamma \rrbracket$  by

$$(e_1 \langle 1 \rangle R e_2 \langle 2 \rangle) = \{ (m_1, m_2) \in \llbracket \Gamma \rrbracket \mid \llbracket \Gamma \vdash e_1 \rrbracket(m_1) R \llbracket \Gamma \vdash e_2 \rrbracket(m_2) \}.$$

## 3.3 Proof Rules

We mainly refer the proof rules of apRHL from [2,16], but we modify the [comp] and [frame] rules to verify differential privacy for continuous random samplings.

$$\begin{array}{c}
x_1 : \tau_1, x_2 : \tau_2 \in \Gamma \quad \Gamma \vdash^t e_1 : \tau_1 \quad \Gamma \vdash^t e_2 : \tau_2 \\
\frac{(\rho_{(x_1 : \tau_1, \Gamma)} \circ \langle \llbracket e_1 \rrbracket, \text{id} \rangle, \rho_{(x_2 : \tau_2, \Gamma)} \circ \langle \llbracket e_2 \rrbracket, \text{id} \rangle) : \Psi \rightarrow \Phi}{\models x_1 \leftarrow e_1 \sim_{(1,0)} x_2 \leftarrow e_2 : \Psi \Rightarrow \Phi} [\text{assn}] \\
\\
\Gamma \vdash^t e_1^1 : \tau_1 \dots \Gamma \vdash^t e_m^1 : \tau_m \quad \Gamma \vdash^t e_1^2 : \tau_1 \dots \Gamma \vdash^t e_m^2 : \tau_m \quad x_1 : \tau, x_2 : \tau \in \Gamma \\
(\langle \llbracket e_1^1 \rrbracket, \dots, \llbracket e_m^1 \rrbracket \rangle, \langle \llbracket e_1^2 \rrbracket, \dots, \llbracket e_m^2 \rrbracket \rangle) : \Psi' \rightarrow \Psi \text{ in } \mathbf{BRel}(\mathbf{Meas}) \\
d : (\tau_1, \dots, \tau_m) \rightarrow \tau \quad (\llbracket d \rrbracket, \llbracket d \rrbracket) : \Psi \rightarrow \overline{\mathcal{G}^{(\gamma, \delta)}}(\text{Eq}_{[\tau]}) \text{ in } \mathbf{BRel}(\mathbf{Meas}) \\
\hline
\models x_1 \stackrel{\$}{\leftarrow} d(e_1^1, \dots, e_m^1) \sim_{(\gamma, \delta)} x_2 \stackrel{\$}{\leftarrow} d(e_1^2, \dots, e_m^2) : \Psi' \Rightarrow (x_1 \langle 1 \rangle = x_2 \langle 1 \rangle) \\
\\
\frac{\models c_1 \sim_{(\gamma, \delta)} c_2 : \Psi \Rightarrow \Phi' \quad \models c'_1 \sim_{(\gamma', \delta')} c'_2 : \Phi' \Rightarrow \Phi}{\models c_1; c'_1 \sim_{(\gamma\gamma', \delta+\delta')} c_2; c'_2 : \Psi \Rightarrow \Phi} [\text{seq}] \quad \frac{}{\models \text{skip} \sim_{(1,0)} \text{skip} : \Phi \Rightarrow \Phi} [\text{skip}] \\
\\
\Gamma \vdash^t b : \text{bool} \quad \Gamma \vdash^t b' : \text{bool} \quad \Psi \Rightarrow b \langle 1 \rangle = b' \langle 2 \rangle \\
\frac{\models c_1 \sim_{(\gamma, \delta)} c'_1 : \Psi \wedge b \langle 1 \rangle \Rightarrow \Phi \quad \models c_2 \sim_{(\gamma, \delta)} c'_2 : \Psi \wedge \neg b \langle 1 \rangle \Rightarrow \Phi}{\models \text{if } b \text{ then } c_1 \text{ else } c_2 \sim_{(\gamma, \delta)} \text{if } b' \text{ then } c'_1 \text{ else } c'_2 : \Psi \Rightarrow \Phi} [\text{cond}] \\
\\
\Gamma \vdash^t e : \text{int} \quad \gamma = \prod_{k=0}^{n-1} \gamma_k \quad \delta = \sum_{k=0}^{n-1} \delta_k \\
\Theta \Rightarrow b_1 \langle 1 \rangle = b_2 \langle 2 \rangle \quad \Theta \wedge e \langle 1 \rangle \geq n \Rightarrow \neg b_1 \langle 1 \rangle \\
\forall k : \text{int}. \models c_1 \sim_{(\gamma_k, \delta_k)} c_2 : \Theta \wedge e \langle 1 \rangle = k \wedge e \langle 1 \rangle \leq n \Rightarrow \Theta \wedge e \langle 1 \rangle > k \\
\hline
\models \text{while } b \text{ do } c_1 \sim_{(\gamma, \delta)} \text{while } b' \text{ do } c_2 : \Theta \wedge b_1 \langle 1 \rangle \wedge e \langle 1 \rangle \geq 0 \Rightarrow \Theta \wedge \neg b_1 \langle 1 \rangle \\
\\
\frac{\models c_1 \sim_{(\gamma, \delta)} c_2 : \Psi \wedge \Theta \Rightarrow \Phi \quad \models c_1 \sim_{(\gamma, \delta)} c_2 : \Psi \wedge \neg \Theta \Rightarrow \Phi}{\models c_1 \sim_{(\gamma, \delta)} c_2 : \Psi \Rightarrow \Phi} [\text{case}] \\
\\
\frac{\models c_1 \sim_{(\gamma, \delta)} c_2 : \Psi \Rightarrow \Phi \quad \Psi' \Rightarrow \Psi \quad \Phi \Rightarrow \Phi'}{\models c_1 \sim_{(\gamma, \delta)} c_2 : \Psi' \Rightarrow \Phi'} [\text{weak}] \quad \frac{\models c_1 \sim_{(\gamma, \delta)} c_2 : \Psi \Rightarrow \Phi}{\models c_2 \sim_{(\gamma, \delta)} c_1 : \Psi^\Phi \Rightarrow \Phi^\Phi} [\text{op}]
\end{array}$$

The relational lifting  $\overline{\mathcal{G}^{(\gamma, \delta)}}$  does not preserve every relation composition. However, it preserve the composition of relations if the relations are *measurable*, that is, the images and inverse images along them of mesurable sets are also measurable (see also [12, Section 3.3]). Generally speaking, it is difficult to check measurability of relations, hence the continuous apRHL is weak for dealing with relation compositions. However, we have the following two special cases:

- The *equality/diagonal* relation on any space is a measurable relation.
- Any relation between *discrete* spaces is automatically a measurable relation.

Hence, the following [comp] rule is an extension of the original [comp] rule in [2]:

$$\begin{array}{c} \Phi \text{ and } \Phi' \text{ are measurable relations} \\ \hline \frac{\models c_1 \sim_{(\gamma, \delta)} c_2 : \Psi \Rightarrow \Phi \quad \models c_2 \sim_{(\gamma', \delta')} c_3 : \Psi' \Rightarrow \Phi'}{\models c_1 \sim_{(\gamma\gamma', \min(\delta+\gamma\delta', \delta'+\gamma'\delta))} c_3 : \Psi \circ \Psi' \Rightarrow \Phi \circ \Phi'} \text{ [comp]} \end{array}$$

To define the [frame] rule in continuous apRHL, for any relation  $\Theta$  on  $\llbracket \Gamma \rrbracket$ , we define the following relation  $\text{Range}(\Theta)$ :

$$\begin{aligned} & \text{Range}(\Theta) \\ &= \{ (\nu_1, \nu_2) \mid \exists A, B \in \Sigma_{\llbracket \Gamma \rrbracket}. (A \times B \subseteq \Theta \wedge \nu_1(A) = \nu_1(\llbracket \Gamma \rrbracket) \wedge \nu_2(B) = \nu_2(\llbracket \Gamma \rrbracket)) \}. \end{aligned}$$

We define the [frame] rule with the construction  $\text{Range}(-)$ :

$$\frac{\models c_1 \sim_{(\gamma, \delta)} c_2 : \Psi \Rightarrow \Phi \quad (\llbracket c_1 \rrbracket, \llbracket c_2 \rrbracket) : \Theta \rightarrow \text{Range}(\Theta)}{\models c_1 \sim_{(\gamma, \delta)} c_2 : \Psi \wedge \Theta \Rightarrow \Phi \wedge \Theta} \text{ [frame]}$$

If  $\llbracket \Gamma \rrbracket$  is countable discrete then the condition  $(\nu_1, \nu_2) \in \text{Range}(\Theta)$  is equivalent to  $\text{supp}(\nu_1) \times \text{supp}(\nu_2) \subseteq \Theta$ , and hence the above [frame] rule is an extension of the original [frame] rule in [2].

Note that if the  $\sigma$ -algebra of the space  $\llbracket \tau \rrbracket$  contains all singleton subsets, and  $\Theta$  does not restrict any variables in  $FV(c_1) \cup FV(c_2)$  then  $(\llbracket c_1 \rrbracket, \llbracket c_2 \rrbracket) : \Theta \rightarrow \text{Range}(\Theta)$ .

### 3.4 Soundness

The soundness of the [assn] and [case] are obtained from the composition of arrows in **BRel(Meas)**. The rule [skip] and [seq] are sound because  $\overline{\mathcal{G}}^{(\gamma, \delta)}$  is the graded relational lifting of  $\mathcal{G} \times \mathcal{G}$  along the forgetful functor  $U : \mathbf{BRel}(\mathbf{Meas}) \rightarrow \mathbf{Meas}^2$ . The rules [weak] and [op] are sound because  $\overline{\mathcal{G}}^{(\gamma, \delta)}$  is monotone with respect to the inclusion order of relations, and preserves opposites of relations. The soundness of [rand] is proved from Fubini theorem. The soundness of [cond] is proved by case analyses. The soundness of [while] is obtained from  $\omega\mathbf{CPO}_\perp$ -enrichment structure of **SRel**. The soundness of [comp] is given by using the measurability of the postconditions. Finally, the [frame] rule is proved from the structure of  $\text{Range}(\Theta)$ .

### 3.5 Mechanisms

In this part, we give a generic method to construct the rules for random samplings, and by instantiating the method we show the soundness of the proof rules in prior researches: [Lap] for Laplacian mechanism [7], [Exp] for Exponential mechanism [14], [Gauss] for Gaussian mechanism [8, Theorem 3.22, Theorem A.1], and [Cauchy] for the mechanism by Cauchy distributions [15].

Let  $f : X \times Y \rightarrow \mathbb{R}$  be a positive measurable function, and  $\nu$  be a measure over  $Y$ . We define the following function  $f_a : \Sigma_Y \rightarrow [0, 1]$  by

$$f_a(B) = \frac{\int_B f(a, -) d\nu}{\int_Y f(a, -) d\nu}.$$

We remark that the function  $f(a, -): Y \rightarrow \mathbb{R}$  is measurable. If the function is not ‘almost everywhere zero’ and Lebesgue integrable, that is,  $0 < \int_Y f(a, -) d\nu < \infty$  then  $f_a(-)$  is a *probability measure*.

The following proposition, which is an extension of [2, Lemma 7], plays the central role in the construction of sound proof rules for random samplings.

**Proposition 3.2** *Let  $f: X \times Y \rightarrow \mathbb{R}$  be a positive measurable function, and  $\nu$  be a measure over  $Y$ . For all  $a, a' \in X$ ,  $\gamma, \gamma' \geq 1$ ,  $\delta \geq 0$ , and  $Z \in \Sigma_Y$  (window set), if the following three conditions hold then for any  $B \in \Sigma_Y$ , we have  $f_a(B) \leq \gamma\gamma' f_{a'}(B) + \delta$ .*

- (i)  $0 < \frac{1}{\gamma'} \int_Y f(a', -) d\nu \leq \int_Y f(a, -) d\nu < \infty$
- (ii)  $\forall b \in Z. f(a, b) \leq \gamma f(a', b)$ , (iii)  $f_a(Y \setminus Z) \leq \delta$ .

### Laplacian mechanism [7].

We give the function  $f: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$  by  $f(a, b) = \frac{2}{\sigma} \exp(\frac{-|b-a|}{\sigma})$ , where  $\sigma > 0$  is the variance of Laplacian mechanism. We introduce the probabilistic operation  $\text{Lap}_\sigma: \mathbf{real} \rightarrow \mathbf{real}$  with  $\llbracket \text{Lap}_\sigma \rrbracket = f(-)$ , whose measurability is shown from the continuity of the mapping  $a \mapsto \int_\alpha^\beta f(a, x) dx$  ( $\alpha, \beta \in \mathbb{R}$ ).

We show  $(f_{(-)}, f_{(-)}): \{ (a, a') \mid |a - a'| < r \} \rightarrow \overline{\mathcal{G}^{(\exp(\frac{r}{\sigma}), 0)}} \text{Eq}_{\mathbb{R}}$  by instantiating Proposition 3.2 as follows: If  $|a - a'| < r$  then the following parameters satisfy the conditions (i)–(iii):  $\gamma = \exp(r/\sigma)$ ,  $\gamma' = 1$ ,  $\delta = 0$ , the function  $f$ , the Lebesgue measure  $\nu$  over  $\mathbb{R}$ , and the window  $Z = \mathbb{R}$ . This implies  $(f_{(-)}, f_{(-)}): \{ (a, a') \mid |a - a'| < r \} \rightarrow \overline{\mathcal{G}^{(\exp(\frac{r}{\sigma}), 0)}} \text{Eq}_{\mathbb{R}}$  since  $\{ (a, a') \mid |a - a'| < r \}$  and  $\text{Eq}_{\mathbb{R}}$  are symmetric.

From the [rand] rule, the following rule is proved:

$$\frac{\Gamma \vdash^t e_1: \mathbf{real} \quad \Gamma \vdash^t e_2: \mathbf{real} \quad m_1 \Psi m_2 \Rightarrow |\llbracket e_1 \rrbracket m_1 - \llbracket e_2 \rrbracket m_2| < r}{\models x \stackrel{\$}{\leftarrow} \text{Lap}_\sigma(e_1) \sim_{(\exp(\frac{r}{\sigma}), 0)} y \stackrel{\$}{\leftarrow} \text{Lap}_\sigma(e_2): \Psi \Rightarrow x\langle 1 \rangle = y\langle 2 \rangle} [\text{Lap}]$$

### Exponential mechanism [14, Modified].

Let  $D$  be the discrete Euclidian space  $\mathbb{Z}^n$ , and  $(R, \nu)$  be a (positive) measure space. Let  $q: D \times R \rightarrow \mathbb{R}$  be a measurable function such that  $\sup_{b \in R} |q(a, b) - q(a', b)| \leq c \cdot \|a - a'\|_1$  for some  $c > 0$ . Suppose  $0 < \int_R \exp(\varepsilon q(a, -)) d\nu < \infty$  for any  $a \in D$ . We give the function  $f: D \times R \rightarrow \mathbb{R}$  by  $f(a, b) = \exp(\varepsilon q(a, b))$ , where  $\varepsilon > 0$  is a constant. We add the value types  $\mathbf{D}$  and  $\mathbf{R}$  with  $\llbracket \mathbf{D} \rrbracket^\Gamma = D$  and  $\llbracket \mathbf{R} \rrbracket^\Gamma = R$  to pWHILE, and introduce the probabilistic operation  $\text{Exp}_{\langle q, \nu, \varepsilon \rangle}: \mathbf{D} \rightarrow \mathbf{R}$  with  $\llbracket \text{Exp}_{\langle q, \nu, \varepsilon \rangle} \rrbracket = f(-)$ .

We show  $(f_{(-)}, f_{(-)}): \{ (a, a') \mid \|a - a'\|_1 < r \} \rightarrow \overline{\mathcal{G}^{(\exp(2\varepsilon rc), 0)}} \text{Eq}_R$  by instantiating Proposition 3.2 as follows: Suppose  $\|a - a'\|_1 < r$ . The following parameters then satisfy the conditions (i)–(iii):  $\gamma = \gamma' = \exp(\varepsilon rc)$ ,  $\delta = 0$ , the function  $f$ , the given measure  $\nu$ , and the window  $Z = R$ .

From the [rand] rule, the following rule is proved:

$$\frac{\Gamma \vdash^t e_1: \mathbf{D} \quad \Gamma \vdash^t e_2: \mathbf{D} \quad m_1 \Psi m_2 \Rightarrow |\llbracket e_1 \rrbracket m_1 - \llbracket e_2 \rrbracket m_2|_1 < r}{\models x \stackrel{\$}{\leftarrow} \text{Exp}_{\langle q, \nu, \varepsilon \rangle}(e_1) \sim_{(\exp(2\varepsilon rc), 0)} y \stackrel{\$}{\leftarrow} \text{Exp}_{\langle q, \nu, \varepsilon \rangle}(e_2): \Psi \Rightarrow x\langle 1 \rangle = y\langle 2 \rangle} [\text{Exp}]$$

**Gaussian mechanism [8, Theorem 3.22, Theorem A.1].**

We give the function  $f: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$  by  $f(a, b) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(b-a)^2}{2\sigma^2})$ , where  $\sigma > 0$  is the variance of Gaussian mechanism. We introduce the probabilistic operation  $\text{Gauss}_\sigma: \mathbf{real} \rightarrow \mathbf{real}$  with  $\llbracket \text{Gauss}_\sigma \rrbracket = f_{(-)}$ , whose continuity is easily proved.

We obtain  $(f_{(-)}, f_{(-)}): \{ (a, a') \mid |a - a'| < r \} \rightarrow \overline{\mathcal{G}^{(\gamma, \delta)}} \text{Eq}_{\mathbb{R}}$  by instantiating Proposition 3.2 as follows: If  $|a - a'| < r$ ,  $1 < \gamma < \exp(1)$ , and  $\gamma' = 1$  hold, and there is  $(3/2) < c$  such that  $2 \log(1.25/\delta) \leq c^2$  and  $(cr/\log \gamma) \leq \sigma$ , then the parameters  $\gamma$ ,  $\gamma'$ , and  $\delta$ , the function  $f$ , and the Lebesgue measure  $\nu$  over  $\mathbb{R}$  satisfy the conditions (i)–(iii) for the window  $Z = \{ b \mid |b - (a + a')/2| \leq (\sigma^2 \log \gamma/r) \}$ .

From the [rand] rule, we obtain the following rule:

$$\frac{\begin{array}{l} \exists c > \frac{3}{2}. (2 \log(\frac{1.25}{\delta}) < c^2 \wedge \frac{cr}{\gamma} \leq \sigma) \quad 1 < \gamma < \exp(1) \\ \Gamma \vdash^t e_1: \mathbf{real} \quad \Gamma \vdash^t e_2: \mathbf{real} \quad m_1 \Psi m_2 \Rightarrow |\llbracket e_1 \rrbracket m_1 - \llbracket e_2 \rrbracket m_2| < r \end{array}}{\vdash x \stackrel{\$}{\leftarrow} \text{Gauss}_\sigma(e_1) \sim_{(\gamma, \delta)} y \stackrel{\$}{\leftarrow} \text{Gauss}_\sigma(e_2): \Psi \Rightarrow x\langle 1 \rangle = y\langle 2 \rangle} [\text{Gauss}]$$

We can relax the above conditions for  $c$  to  $((1 + \sqrt{3})/2) < c$  and  $2 \log(0.66/\delta) < c^2$  by changing the window  $Z$  to  $\{ b \mid b \leq (a + a')/2 + (\sigma^2 \log \gamma/r) \}$  when  $a \leq a'$  and  $\{ b \mid b \geq (a + a')/2 - (\sigma^2 \log \gamma/r) \}$  when  $a' \leq a$ .

**Mechanism of Cauchy distributions [15]**

We give the function  $f: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$  by  $f(a, b) = \frac{\rho}{\pi((a-b)^2 + \rho^2)}$ . We introduce the probabilistic operation  $\text{Cauchy}_\rho: \mathbf{real} \rightarrow \mathbf{real}$  with  $\llbracket \text{Cauchy}_\rho(e) \rrbracket^\Gamma m = f_{(-)}$ , whose continuity is easily proved.

Let  $\gamma = 1 + \frac{r^2 + r\sqrt{r^2 + 4\rho^2}}{2\rho^2}$ . We obtain  $(f_{(-)}, f_{(-)}): \{ (a, a') \mid |a - a'| < r \} \rightarrow \overline{\mathcal{G}^{(\gamma, 0)}} \text{Eq}_{\mathbb{R}}$  by instantiating Proposition 3.2 as follows: If  $|a - a'| < r$  then the parameters satisfy the conditions (i)–(iii):  $\gamma$ ,  $\gamma' = 1$ ,  $\delta = 0$ , the Lebesgue measure  $\nu$  over  $\mathbb{R}$ , and the window  $Z = \mathbb{R}$ .

From the [rand] rule, we obtain the following rule:

$$\frac{\Gamma \vdash^t e: \mathbf{real} \quad m_1 \Psi m_2 \Rightarrow |\llbracket e_1 \rrbracket m_1 - \llbracket e_2 \rrbracket m_2| < r}{\vdash x \stackrel{\$}{\leftarrow} \text{Cauchy}_\rho(e_1) \sim_{(\gamma, 0)} y \stackrel{\$}{\leftarrow} \text{Cauchy}_\rho(e_2): \Psi \Rightarrow (\pi_x \times \pi_y)^{-1}(\text{Eq}_{\mathbb{R}})} [\text{Cauchy}]$$

**4 An Example: The Above Threshold Algorithm**

Barthe, Gaboardi, Grégoire, Hsu, and Strub extended the logic apRHL to the logic apRHL+ with new proof rules to describe the *sparse vector technique* (see also [8, Section 3.6]). They gave a formal proof of the differential privacy of *above threshold algorithm* in [1].

In this section, we demonstrate that the above threshold algorithm with *real-valued queries* is proved with *almost the same proof* as in [1]. The new proof rules of apRHL+ are still sound in the framework of the continuous apRHL.

We consider the following algorithm **AboveT**:



**Algorithm 1** The Above Threshold Algorithm ([1], Modified)

---

```

1: AboveT( $T$ : real,  $Q$ : queries,  $d$ : data)
2:    $j \leftarrow 1$ ;  $r \leftarrow |Q| + 1$ ;  $T \stackrel{\$}{\leftarrow} \text{Lap}_{\varepsilon/2}(t)$ ;
3:   while  $j < |Q|$  do
4:      $S \stackrel{\$}{\leftarrow} \text{Lap}_{\varepsilon/4}(\text{eval}(Q, i, d))$ ;
5:     if  $T \leq S \wedge r = |Q| + 1$  then
6:        $r \leftarrow j$ ;
7:        $j \leftarrow j + 1$ 

```

---

We recall the setting of this algorithm. This algorithm has two fixed parameters: the threshold  $t$ : **real** and the set  $Q$ : **queries** of queries where  $|Q|$ : **int** is the number of  $Q$ . The input variable is  $d$ : **int**, and the output variable is  $r$ : **int**. We prepare the new value types **queries** and **data** with  $\llbracket \text{data} \rrbracket = \mathbb{R}^N$  and **queries** = **int** (alias), and the typings  $j$ : **int**,  $T$ : **real**, and  $S$ : **real**. We assume that an operation  $\text{eval}: (\text{queries}, \text{int}, \text{data}) \rightarrow \text{real}$  is given for evaluating  $i$ -th query in  $Q$  for the input  $d$ . We require  $\llbracket \text{eval} \rrbracket$  to be 1-*sensitivity* for the data  $d$ , that is,  $\|d - d'\|_1 \leq 1 \Rightarrow |\llbracket \text{eval} \rrbracket(Q, i, d) - \llbracket \text{eval} \rrbracket(Q, i, d')| \leq 1$ .

The differential privacy of **Above** is characterised as follows:

$$\models \text{AboveT} \sim_{\exp(\varepsilon), 0} \text{AboveT}: \|d\langle 1 \rangle - d\langle 2 \rangle\|_1 \leq 1 \Rightarrow r\langle 1 \rangle = r\langle 2 \rangle.$$

The following rules in apRHL+ are sound in the framework of continuous apRHL:

$$\frac{\forall i: \text{int}. \models c_1 \sim_{(\gamma, \delta_i)} c_2: \Psi \Rightarrow (x\langle 1 \rangle = i \Rightarrow x\langle 2 \rangle = i) \quad \sum_{i: \text{int}} \llbracket \delta_i \rrbracket = \delta}{\models c_1 \sim_{(\gamma, \delta)} c_2: \Psi \Rightarrow x\langle 1 \rangle = x\langle 2 \rangle} \text{ [Forall-Eq]}$$

$$\frac{\Gamma \vdash^t e_1: \text{real} \quad \Gamma \vdash^t e_2: \text{real} \quad m_1 \Psi m_2 \Rightarrow |\llbracket e_1 \rrbracket m_1 + r' - \llbracket e_2 \rrbracket m_2| < r}{\models x \stackrel{\$}{\leftarrow} \text{Lap}_\sigma(e_1) \sim_{(\exp(\frac{r}{\sigma}), 0)} y \stackrel{\$}{\leftarrow} \text{Lap}_\sigma(e_2): \Psi \Rightarrow x\langle 1 \rangle + r' = y\langle 2 \rangle} \text{ [LapGen]}$$

$$\frac{\Gamma \vdash^t e_1: \text{real} \quad \Gamma \vdash^t e_2: \text{real} \quad x \notin FV(e_1) \quad y \notin FV(e_2)}{\models x \stackrel{\$}{\leftarrow} \text{Lap}_\sigma(e_1) \sim_{(1, 0)} y \stackrel{\$}{\leftarrow} \text{Lap}_\sigma(e_2): \Psi \Rightarrow x\langle 1 \rangle - y\langle 2 \rangle = e_1\langle 1 \rangle - e_2\langle 2 \rangle} \text{ [LapNull]}$$

Hence we extend the continuous apRHL by adding these rules, and therefore we construct a formal proof almost the same proof as in [1] in the extended continuous apRHL.

The soundness of the rule [Forall-Eq] is proved from the following lemma:

**Lemma 4.1** ([1, Proposition 6], Modified) *If  $x: \tau$  and the space  $\llbracket \tau \rrbracket$  is countable discrete then*

$$\bigcap_{i \in \llbracket \tau \rrbracket} \mathcal{G}^{(\gamma, \delta_i)}(x\langle 1 \rangle = i \Rightarrow x\langle 2 \rangle = i) \subseteq \mathcal{G}^{(\gamma, \sum_{i \in \llbracket \tau \rrbracket} \delta_i)}(x\langle 1 \rangle = x\langle 2 \rangle).$$

The soundness of the rule [LapGen] is proved from the rules [Lap] and [assn] and the semantic equivalence  $\llbracket x \stackrel{\$}{\leftarrow} \text{Lap}_\sigma(e + r'); x \leftarrow x - r' \rrbracket = \llbracket x \stackrel{\$}{\leftarrow} \text{Lap}_\sigma(e) \rrbracket$ . The soundness of [LapNull] is proved by using the [LapGen] and [Frame] rules.

### Formal Proof

We now demonstrate that the  $(\varepsilon, 0)$ -differential privacy of algorithm **AboveT** is proved with almost the same proof as in [1].

From the [Forall-Eq] rule with variable  $r$ , it suffices to prove for all integer  $i$ ,

$$\models \mathbf{AboveT} \sim_{\exp(\varepsilon), 0} \mathbf{AboveT}: \|d\langle 1 \rangle - d\langle 2 \rangle\|_1 \leq 1 \Rightarrow (r\langle 1 \rangle = i \Rightarrow r\langle 2 \rangle = i).$$

We denote by  $c_0$  the sub-command consisting of the initialization line 2 of **AboveT**. From the rules [assn], [LapGen] rule with  $r = r' = 1$ , and  $\sigma = 2/\varepsilon$ , [seq], and [frame] we obtain

$$\models c_0 \sim_{\exp(\varepsilon/2), 0} c_0: \|d\langle 1 \rangle - d\langle 2 \rangle\|_1 \leq 1 \Rightarrow \|d\langle 1 \rangle - d\langle 2 \rangle\|_1 \leq 1 \wedge \Psi.$$

where

$$\Psi = T\langle 1 \rangle + 1 = T\langle 2 \rangle \wedge j\langle 1 \rangle = j\langle 2 \rangle \wedge j\langle 1 \rangle = 1 \wedge r\langle 1 \rangle = r\langle 2 \rangle \wedge r\langle 1 \rangle = |Q| + 1.$$

We denote by  $c_1$  and  $c_2$  the main loop and the body of the main loop respectively (i.e.  $c_1 = \mathbf{while} (j < |Q|) \mathbf{do} c_2$ ). We aim to prove the following judgement by using the [while] rule:

$$\models c_1 \sim_{\exp(\varepsilon/2), 0} c_1: (\|d\langle 1 \rangle - d\langle 2 \rangle\|_1 \leq 1 \wedge \Psi) \Rightarrow (r\langle 1 \rangle = i \Rightarrow r\langle 2 \rangle = i).$$

To prove this, it suffices to show the following cases for the loop body  $c_2$ :

- (i) If  $k < i$  then  $\models c_2 \sim_{1, 0} c_2: (\Theta \wedge j\langle 1 \rangle = k) \Rightarrow (\Theta \wedge j\langle 1 \rangle > k)$
- (ii) If  $k = i$  then  $\models c_2 \sim_{\exp(\varepsilon/2), 0} c_2: (\Theta \wedge j\langle 1 \rangle = k) \Rightarrow (\Theta \wedge j\langle 1 \rangle > k)$
- (iii) If  $k > i$  then  $\models c_2 \sim_{1, 0} c_2: (\Theta \wedge j\langle 1 \rangle = k) \Rightarrow (\Theta \wedge j\langle 1 \rangle > k)$

Here, we provide the following *loop invariant* as follows:

$$\begin{aligned} \Theta = & (j\langle 1 \rangle < i \Rightarrow ((r\langle 1 \rangle = |Q| + 1 \Rightarrow r\langle 2 \rangle = |Q| + 1) \wedge (r\langle 1 \rangle = |Q| + 1 \vee r\langle 1 \rangle < i))) \\ & \wedge (j\langle 1 \rangle \geq i \Rightarrow (r\langle 1 \rangle = i \Rightarrow r\langle 2 \rangle = i)) \\ & \wedge \|d\langle 1 \rangle - d\langle 2 \rangle\|_1 \leq 1 \wedge T\langle 1 \rangle + 1 = T\langle 2 \rangle \wedge j\langle 1 \rangle = j\langle 2 \rangle \end{aligned}$$

The judgement in the case (i) is proved from the rules [seq], [assn], [cond], and [frame] and the following fact obtained from the [LapNull] rule:

$$\begin{aligned} \models S \stackrel{\$}{\leftarrow} \mathbf{Lap}_{\varepsilon/4}(\mathbf{eval}(Q, i, d)) \sim_{1, 0} S \stackrel{\$}{\leftarrow} \mathbf{Lap}_{\varepsilon/4}(\mathbf{eval}(Q, i, d)): \\ (\|d\langle 1 \rangle - d\langle 2 \rangle\|_1 \leq 1) \wedge (T\langle 1 \rangle + 1 = T\langle 2 \rangle) \Rightarrow ((S\langle 1 \rangle < T\langle 1 \rangle) \Rightarrow (S\langle 2 \rangle < T\langle 2 \rangle)). \end{aligned}$$

The case (ii) is proved from the rules [seq], [assn], [cond], and [frame] and the following fact obtained from the [LapGen] rule:

$$\begin{aligned} \models S \stackrel{\$}{\leftarrow} \mathbf{Lap}_{\varepsilon/4}(\mathbf{eval}(Q, i, d)) \sim_{\exp(\varepsilon/2), 0} S \stackrel{\$}{\leftarrow} \mathbf{Lap}_{\varepsilon/4}(\mathbf{eval}(Q, i, d)): \\ (\|d\langle 1 \rangle - d\langle 2 \rangle\|_1 \leq 1 \wedge T\langle 1 \rangle + 1 = T\langle 2 \rangle) \Rightarrow (S\langle 1 \rangle + 1 = S\langle 2 \rangle \wedge T\langle 1 \rangle + 1 = T\langle 2 \rangle). \end{aligned}$$

The case (iii) is proved in the similar way as (i).

## Acknowledgement

The author thanks Shin-ya Katsumata for many valuable comments and stimulating discussions, Marco Gaboardi for helpful suggestions and the introduction of his preprint of [1] in arXiv, Masahito Hasegawa, Naohiko Hoshino, and Takeo Uramoto for advices that contributed to improve the writing of this paper.

## References

- [1] Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Jastin Hsu, and Pierre-Yves Strub. Proving Differential Privacy via Probabilistic Couplings. In *Proceedings of Thirty-First Annual ACM/IEEE Symposium on LOGIC IN COMPUTER SCIENCE (LICS)*, to appear.
- [2] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella-Béguelin. Probabilistic relational reasoning for differential privacy. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, pages 97–110, New York, NY, USA, 2012. ACM.
- [3] Gilles Barthe and Federico Olmedo. Beyond differential privacy: Composition theorems and relational logic for f-divergences between probabilistic programs. In FedorV. Fomin, R?si?? Freivalds, Marta Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming*, volume 7966 of *Lecture Notes in Computer Science*, pages 49–60. Springer Berlin Heidelberg, 2013.
- [4] Nick Benton. Simple relational correctness proofs for static analyses and program transformations. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '04)*, number MSR-TR-2005-26, page 43. ACM, January 2004.
- [5] Daniel Brown and Riccardo Pucella. Categories of timed stochastic relations. *Electronic Notes in Theoretical Computer Science*, 249:193 – 217, 2009. Proceedings of the 25th Conference on Mathematical Foundations of Programming Semantics (MFPS 2009).
- [6] E.P de Vink and J.J.M.M Rutten. Bisimulation for probabilistic transition systems: a coalgebraic approach. *Theoretical Computer Science*, 221(1 - 2):271 – 293, 1999.
- [7] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer Berlin Heidelberg, 2006.
- [8] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3-4):211–407, 2013.
- [9] Michèle Giry. A categorical approach to probability theory. In B. Banaschewski, editor, *Categorical Aspects of Topology and Analysis*, volume 915 of *Lecture Notes in Mathematics*, pages 68–85. Springer Berlin Heidelberg, 1982.
- [10] Bart Jacobs and Jesse Hughes. Simulations in coalgebra. *Electronic Notes in Theoretical Computer Science*, 82(1):128–149, 2003. CMCS'03, Coalgebraic Methods in Computer Science (Satellite Event for ETAPS 2003).
- [11] Shin-ya Katsumata. Parametric effect monads and semantics of effect systems. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, pages 633–645, New York, NY, USA, 2014. ACM.
- [12] Shin-ya Katsumata and Tetsuya Sato. Codensity Liftings of Monads. In Lawrence S. Moss and Pawel Sobocinski, editors, *6th Conference on Algebra and Coalgebra in Computer Science (CALCO 2015)*, volume 35 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 156–170, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [13] Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
- [14] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '07, pages 94–103, Washington, DC, USA, 2007. IEEE Computer Society.
- [15] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 75–84, New York, NY, USA, 2007. ACM.
- [16] Federico Olmedo. *Approximate Relational Reasoning for Probabilistic Programs*. PhD thesis, Technical University of Madrid, 2014.
- [17] Prakash Panangaden. The category of markov kernels. *Electronic Notes in Theoretical Computer Science*, 22:171 – 187, 1999. PROBMIV'98, First International Workshop on Probabilistic Methods in Verification.

# Coalgebraic minimization of automata by initiality and finality

Jurriaan Rot

*Université de Lyon, ENS de Lyon, CNRS, UCB Lyon 1, LIP*

---

## Abstract

Deterministic automata can be minimized by partition refinement (Moore's algorithm, Hopcroft's algorithm) or by reversal and determinization (Brzozowski's algorithm). In the coalgebraic perspective, the first approach can be phrased in terms of a minimization construction along the final sequence of a functor, whereas a crucial part of the second approach is based on a reachability construction along the initial sequence of another functor. We employ this coalgebraic perspective to establish a precise relationship between the two approaches to minimization, and show how they can be combined. Part of these results are extended to an approach for language equivalence of a general class of systems with branching, such as non-deterministic automata.

*Keywords:* minimization, automata, coalgebra

---

## 1 Introduction

The problem of minimizing deterministic automata has been studied since the early days of automata theory, and a number of different approaches have been proposed. Probably the most well-known family of algorithms, which includes Hopcroft's [11] and Moore's algorithm [19] as well as typical textbook constructions [12], is based on a stepwise refinement of a partition of states. Another approach, due to Brzozowski [7], is based on determinization and reversal. That approach appears (and is usually considered) to be fundamentally different than partition refinement [3,24]. To the best of our knowledge, a connection was only established in the work of Champarnaud et al [8] (and further extended in [9]), who explicitly showed how the partition of states that are language equivalent is obtained from the reversed determinized automaton that appears in Brzozowski's algorithm.

Partition refinement can be phrased abstractly as an inductive computation along the final sequence of a functor, generalizing from automata to coalgebras [1].

---

<sup>1</sup> This work was performed within the framework of the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR), and was supported by project ANR 12IS02001 PACE.

<sup>2</sup> Email: [jurriaan.rot@ens-lyon.fr](mailto:jurriaan.rot@ens-lyon.fr)

Starting with [6], Brzowski's algorithm has also received significant attention from a coalgebraic perspective, as an elegant instance of duality between algebra and coalgebra [5], in several different formulations [5,4,15].

In this paper we employ the coalgebraic perspective on partition refinement and Brzowski's algorithm to understand and establish their relationship. First, we dualize the construction of [1] and combine it with a variant of the Brzowski construction from [15], to obtain a minimization construction based on a stepwise computation of reachability along an initial sequence. We then show how the  $i$ -th step of this reachability construction yields the  $i$ -th partition of states in partition refinement by a simple factorization, thus establishing a fundamental connection between the two minimization constructions. Based on this result, we define a minimization construction that combines partition refinement with the computation of reachability. In our motivating example of deterministic automata, we retrieve the combined minimization construction due to Champarnaud et al [8].

Our Brzowski construction is based on [15], where it is formulated for systems with branching, such as non-deterministic, alternating and tree automata. In the last part of the paper, we consider such branching systems, and show how the reachability computation yields an abstract procedure for language equivalence.

*Outline.* In Section 2 we describe the ideas of partition refinement and Brzowski's algorithm and their connection, for deterministic automata. Section 3 contains preliminaries, Section 4 recalls coalgebraic partition refinement, and Section 5 introduces the dual reachability construction. Section 6 introduces the abstract Brzowski construction, and Section 7 establishes the connection with partition refinement. Section 8 concerns branching systems. Proofs can be found in the appendix.

*Acknowledgments.* The author is grateful to Filippo Bonchi, Matias Lee, Damien Pous and Alexandra Silva for comments, suggestions and discussions.

## 2 Minimization of deterministic automata

We fix an alphabet  $A$ , denote the set of words over  $A$  by  $A^*$  and the empty word by  $\varepsilon$ . A *deterministic automaton* is a triple  $(X, o, f)$  consisting of a set of states  $X$ , a transition function  $f: X \rightarrow X^A$  and an output function  $o: X \rightarrow 2$ , where  $2 = \{0, 1\}$  is a two-element set. Note that the state space  $X$  is not required to be finite, and there is no initial state. The semantics of an automaton is a function  $l: X \rightarrow 2^{A^*}$  mapping each state to the language it accepts, inductively defined by  $\varepsilon \in l(x)$  iff  $o(x) = 1$  and  $aw \in l(x)$  iff  $w \in l(f(x)(a))$ , for any letter  $a \in A$  and word  $w \in A^*$ .

Our aim is to *minimize* deterministic automata: given an automaton  $(X, o, f)$  we search the automaton with the least number of states that accepts the same languages as those accepted by the states in  $X$ . Formulated slightly more abstractly, the aim is to find a factorization of the semantics  $l: X \rightarrow 2^{A^*}$  as a surjective function  $e: X \rightarrow E$  followed by an injective function  $m: E \rightarrow 2^{A^*}$ . Such a factorization uniquely turns the set  $E$  into a (minimal) automaton accepting all languages of states in  $X$ . We describe the ideas underlying two standard approaches to minimization, based respectively on representing  $E$  as a quotient of states, and on representing the image of  $X$  along  $l$  by taking a quotient of words.

### 2.1 Minimization by equivalence of states

Let  $(X, o, f)$  be a deterministic automaton, with language semantics  $l: X \rightarrow 2^{A^*}$ . Consider the equivalence relation  $\equiv \subseteq X \times X$  defined as the kernel of  $l$ , i.e.,  $x \equiv y$  iff  $l(x) = l(y)$ . Two states are related by  $\equiv$  precisely if they are language equivalent. Once we computed the relation  $\equiv$ , the minimization of our automaton can be obtained as the quotient of states w.r.t.  $\equiv$ .

The relation  $\equiv$  can be *approximated* by defining a family of equivalence relations  $\equiv_n \subseteq X \times X$  indexed by natural numbers, called Moore equivalences [3], as follows:  $x \equiv_n y$  iff  $\forall w \in A^*$  with  $|w| < n$ : ( $w \in l(x)$  iff  $w \in l(y)$ ), where  $|w|$  is the length of a word  $w$ . In words,  $\equiv_n$  is language equivalence for words with length below  $n$ . The point is that we can characterize  $\equiv_n$  by induction, setting  $\equiv_0 = X \times X$  and

$$x \equiv_{n+1} y \quad \text{iff} \quad o(x) = o(y) \text{ and } \forall a \in A : f(x)(a) \equiv_n f(y)(a).$$

If  $X$  is finite, then this inductive computation will eventually stabilize, at which point we have computed the relation  $\equiv$  and, hence, a minimal automaton (e.g., [12]). (The usual presentation is slightly different, starting from the relation that distinguishes between accepting and non-accepting states, and leaving the condition  $o(x) = o(y)$  out. We prefer the above variation to match the theory in Section 4.)

Phrasing the above inductive characterization in terms of partitions of  $X$  yields a construction based on stepwise refinement of partitions. Moore's minimization algorithm [19], for instance, is an implementation of this construction, whereas Hopcroft's minimization algorithm [11] is a more advanced (and efficient) version of partition refinement. We refer to [3] for a detailed analysis of these algorithms.

### 2.2 Minimization by equivalence of words

We define an equivalence relation  $\approx \subseteq A^* \times A^*$  by  $w \approx v$  iff  $\forall x \in X : (w \in l(x) \text{ iff } v \in l(x))$ . This relation is dual to  $\equiv$ , in the sense that it is the kernel of the *transpose*  $l^b: A^* \rightarrow 2^X$  of the language semantics  $l$ . Two words are related by  $\approx$  if there is no state in the automaton that accepts one but not the other.

Given an equivalence class  $[w]$  in the quotient  $A^*/\approx$ , a state  $x \in X$  either accepts all words in  $[w]$ , or none. Hence, the language of every  $x \in X$  arises as a union  $\bigcup \{[w] \mid w \in l(x)\}$  of equivalence classes in  $A^*/\approx$ . The set  $\{\{[w] \mid w \in l(x)\} \mid x \in X\}$  is isomorphic to the set of languages accepted by the automaton (the image of  $X$  along  $l$ ), which is (the state space of) a minimal automaton.

But how are these equivalence classes of words computed and represented? The crux is that there is an isomorphism between the quotient  $A^*/\approx$  and the set  $R = \{\{x \in X \mid w \in l(x)\} \mid w \in A^*\}$ , that is, every equivalence class of words is represented as the set of states accepting these words. The set  $R$  has an inductive characterization, as the limit of:

$$R_0 = \emptyset \quad R_{i+1} = \{\{x \in X \mid f(x)(a) \in S\} \mid a \in A, S \in R_i\} \cup \{\{x \in X \mid o(x) = 1\}\}$$

If the state space  $X$  is finite, then this sequence stabilizes after a finite number of steps, at which point we computed  $R$  and, hence, the partition of  $A^*$ . The language of a state  $x \in X$  is then represented by the set  $\{S \in R \mid x \in S\}$ , and (the state

space of) our minimal automaton is obtained by taking  $\{\{S \in R \mid x \in S\} \mid x \in X\}$ . Similar to the case of  $\equiv_i$ , the above presentation of the sets  $R_i$  is chosen to match the abstract theory of Section 6.

The inductive computation of  $R_i$ 's corresponds to the *reachable* (sets of) states in the automaton with state space  $2^X$  obtained from  $(X, o, f)$  by reversing transitions, turning the set of final states into the initial state and determinizing. This computation is at the heart of *Brzowski's minimization algorithm* [7]. That algorithm minimizes a deterministic automaton (with initial and final states) by doing the following twice: reverse and determinize the automaton, and take the part that is reachable from the new initial state.

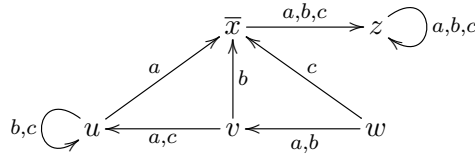
Brzowski's algorithm is usually explained differently, based on the fact that the reverse of an automaton recognizes the reverse language (e.g., [22,3,5]). We prefer the above explanation in terms of equivalence classes, because it explains the construction directly in terms of the original automaton, and highlights a tight correspondence between Brzowski's construction and partition refinement.

Indeed, for each  $i$  we have:

$$x \equiv_i y \quad \text{iff} \quad (\forall S \in R_i : x \in S \text{ iff } y \in S) \quad (1)$$

which means that  $\equiv_i$  can be obtained directly from  $R_i$  and, as shown in [8], that  $\equiv$  can be obtained from  $R$ . In terms of partitions, writing  $E_i$  for the quotient of  $X$  by  $\equiv_i$ , the above equation (1) shows how to compute  $E_i$  from  $R_i$  by *splitting* the set  $X$  according to the sets in  $R_i$ : informally,  $E_i$  is obtained by starting with the trivial partition  $E_i = \{X\}$  and then, for each  $S \in R_i$ , replacing each  $Q \in E_i$  by  $Q \setminus S$  and  $Q \cap S$  if both are nonempty. It is not difficult to see that to compute  $E_{i+1}$  from  $E_i$ , one only needs to compute  $R_{i+1}$  from  $R_i$  and split all the equivalence classes in  $E_i$  according to the new sets (splitters) in  $R_{i+1}$ . This is the basis of an algorithm, proposed in [8], that combines partition refinement with Brzowski's algorithm.

**Example 2.1** Consider the following deterministic automaton over the alphabet  $\{a, b, c\}$ , where the only accepting state is  $x$ .



We compute the quotients  $E_i$  of  $X$  by  $\equiv_i$ , and the sets  $R_i$ 's as explained above:

$$\begin{aligned}
 E_0 &= \{\{x, u, v, w, z\}\} & R_0 &= \emptyset \\
 E_1 &= \{\{x\}, \{u, v, w, z\}\} & R_1 &= \{\{x\}\} \\
 E_2 &= \{\{x\}, \{u\}, \{v\}, \{w\}, \{z\}\} & R_2 &= \{\{x\}, \{u\}, \{v\}, \{w\}\} \\
 & & R_3 &= \{\{x\}, \{u\}, \{v\}, \{w\}, \{u, v\}, \emptyset\} \\
 & & R_4 &= \{\{x\}, \{u\}, \{v\}, \{w\}, \{u, v\}, \emptyset, \{u, w\}, \{v, w\}\}
 \end{aligned}$$



Each  $E_i$  is computed from  $R_i$  by only identifying those states that appear in the same sets in  $R_i$ , or, more efficiently, by splitting the partitions in  $E_{i-1}$  according to the newly added sets in  $R_i$ . For instance, we obtain  $E_2$  from  $E_1$  and  $R_2$  by splitting  $\{x\}$  and  $\{u, v, w, z\}$  by  $\{u\}, \{v\}, \{w\}$ , in particular by splitting  $\{u, v, w, z\}$  by  $\{u\}$ , yielding  $\{u\}, \{v, w, z\}$ ; then  $\{v, w, z\}$  by  $\{v\}$  yielding  $\{v\}, \{w, z\}$ ; and finally  $\{w, z\}$  by  $\{w\}$  (notice that the order of splitting does not matter). Observe that we can compute  $E_i$  from  $R_i$ , but not vice versa. And the sequence of  $E_i$ 's may stabilize earlier than the sequence of  $R_i$ 's.

### 3 Preliminaries

For the remainder of this paper, we assume familiarity with basic notions of category theory. Given a category  $\mathbf{C}$ , a *coalgebra* for a functor  $B: \mathbf{C} \rightarrow \mathbf{C}$  is a pair  $(X, c)$  where  $X$  is an object in  $\mathbf{C}$  and  $c$  is a morphism  $c: X \rightarrow BX$ . A coalgebra homomorphism from  $(X, c)$  to  $(Y, d)$  is a  $\mathbf{C}$ -morphism  $h: X \rightarrow Y$  such that  $Fh \circ c = d \circ h$ . The category of coalgebras for a functor  $B$  is denoted by  $\text{coalg}(B)$ . A coalgebra  $(Z, \zeta)$  is called *final* if it is a final object in  $\text{coalg}(B)$ , i.e., for every coalgebra  $(X, c)$  there exists a unique coalgebra morphism from  $(X, c)$  to  $(Z, \zeta)$ .

For our running example, consider the functor  $B: \mathbf{Set} \rightarrow \mathbf{Set}$  defined by  $BX = 2 \times X^A$ , where  $A$  is a fixed set. A  $B$ -coalgebra  $\langle o, f \rangle: X \rightarrow 2 \times X^A$  is a deterministic automaton (with no initial state), as in Section 2. The functor  $B$  has a final coalgebra, given by the set of languages over  $A$ . The unique morphism from any automaton to this final coalgebra maps each state to the language it accepts [21].

An *algebra* for a functor  $L: \mathbf{D} \rightarrow \mathbf{D}$  is defined dually to a coalgebra, i.e., it is a pair  $(X, a)$  where  $a: LX \rightarrow X$ , and an algebra morphism from  $(X, a)$  to  $(Y, b)$  is a morphism  $h: X \rightarrow Y$  such that  $h \circ a = b \circ Lh$ . The category of  $L$ -algebras is denoted by  $\text{alg}(L)$ . An algebra is called *initial* if it is an initial object in  $\text{alg}(L)$ .

As an example, consider the functor  $L: \mathbf{Set} \rightarrow \mathbf{Set}$  defined by  $LX = A \times X + 1$ , where  $A$  is a fixed set and  $1 = \{*\}$  a singleton. An  $L$ -algebra consists of a set  $X$  and a map  $[g, \iota]: A \times X + 1 \rightarrow X$ . We interpret  $L$ -algebras as deterministic automata with initial state  $\iota(*)$  and transition function  $g$  (but no final states). This functor  $L$  has an initial algebra, given by the set of words  $A^*$  with the empty word  $\varepsilon$  as initial state and  $(a, w) \in A \times A^*$  mapped to the concatenation  $aw$ . Given an  $L$ -algebra (deterministic automaton), the unique morphism from  $A^*$  maps a word  $w$  to the state that is reached after processing  $w$  from the initial state, reading the letters from right to left.

*Contravariant adjunctions.* We will consider functors  $F: \mathbf{C}^{\text{op}} \rightarrow \mathbf{D}$ ,  $G: \mathbf{D}^{\text{op}} \rightarrow \mathbf{C}$  that form an adjunction  $F^{\text{op}} \dashv G$ , i.e., such that there is a natural bijection  $\mathbf{C}(X, GY) \cong \mathbf{D}(Y, FX)$ . We denote both sides of this bijection by  $(-)^b$ , and for a morphism  $f$  in either of the two homsets we call  $f^b$  the *transpose* of  $f$ . An adjunction as above has two units  $\eta: \text{Id} \Rightarrow GF$  and  $\iota: \text{Id} \Rightarrow FG$ . For a morphism  $f: X \rightarrow GY$  the transpose is given by  $f^b = Ff \circ \iota_Y$  and, for  $g: Y \rightarrow FX$ , by  $g^b = Gg \circ \eta_X$ . The standard example is  $\mathbf{C} = \mathbf{D} = \mathbf{Set}$  with  $F = G = 2^-$  the contravariant powerset functor.

To avoid too much of the  $(-)^{\text{op}}$  notation, we treat  $F$  and  $G$  as *contravariant functors* between  $\mathbf{C}$  and  $\mathbf{D}$ , meaning that they reverse the direction of arrows, and refer to an adjunction as above as a *contravariant adjunction*. This should not lead



to confusion, as all the adjunctions considered in this paper are contravariant.

*Factorization systems.* Let  $\mathbf{C}$  be a category, and  $\mathcal{E}, \mathcal{M}$  classes of morphisms in  $\mathbf{C}$ . The pair  $(\mathcal{E}, \mathcal{M})$  is called a *factorization system* if (a) both  $\mathcal{E}$  and  $\mathcal{M}$  are closed under isomorphisms, (b) for every morphism  $f$  in  $\mathbf{C}$  there is an  $(\mathcal{E}, \mathcal{M})$ -factorization: a pair of morphisms  $e \in \mathcal{E}$ ,  $m \in \mathcal{M}$  s.t.  $m \circ e = f$ , and (c) for every commutative square as on the left-hand side of (2), with  $e \in \mathcal{E}$  and  $m \in \mathcal{M}$ , there is a unique diagonal  $d$  making the right-hand side commute [2].

$$\begin{array}{ccc} A & \xrightarrow{e} & B \\ g \downarrow & & \downarrow f \\ C & \xrightarrow{m} & D \end{array} \quad \begin{array}{ccc} A & \xrightarrow{e} & B \\ g \downarrow & \searrow d & \downarrow f \\ C & \xrightarrow{m} & D \end{array} \quad (2)$$

Both  $\mathcal{E}$  and  $\mathcal{M}$  are closed under composition of morphisms. Further,  $(\mathcal{E}, \mathcal{M})$ -factorizations are unique up to isomorphism [2]. We denote morphisms in  $\mathcal{E}$  by arrows of the form  $A \twoheadrightarrow B$  and morphisms in  $\mathcal{M}$  by arrows of the form  $C \rightarrowtail D$ . If  $\mathcal{E}$  is the class of epimorphisms and  $\mathcal{M}$  the class of monomorphisms then we speak of an (epi,mono)-factorization system. A standard example is the (epi,mono)-factorization system of the category **Set** of sets and functions.

Given a functor  $F: \mathbf{C} \rightarrow \mathbf{C}$  on a category  $\mathbf{C}$  with a factorization system  $(\mathcal{E}, \mathcal{M})$ , if  $F$  preserves morphisms in  $\mathcal{M}$  then the factorization system lifts to  $\mathbf{coalg}(F)$  [18,1]. If  $F$  preserves morphisms in  $\mathcal{E}$  then the factorization system lifts to  $\mathbf{alg}(F)$ . A category  $\mathbf{C}$  is called *wellpowered* if, for every object  $X$ , there is (up to isomorphism) only a set of monomorphisms with codomain  $X$ . It is called *cowellpowered* if every object  $X$  has (up to isomorphism) only a set of epimorphisms with domain  $X$ .

## 4 Minimization

In this section we recall from [1] the notion of minimization, and an associated abstract partition refinement procedure. Throughout this section, let  $\mathbf{C}$  be a complete category with an  $(\mathcal{E}, \mathcal{M})$ -factorization system, and  $B: \mathbf{C} \rightarrow \mathbf{C}$  a functor.

**Definition 4.1** A *minimization* of a  $B$ -coalgebra  $(X, c)$  is a  $B$ -coalgebra  $(E, \epsilon)$  with a coalgebra morphism  $e: (X, c) \rightarrow (E, \epsilon)$  with  $e \in \mathcal{E}$  such that for every coalgebra morphism  $e': (X, c) \rightarrow (Y, d)$  with  $e' \in \mathcal{E}$  there is a unique coalgebra morphism  $h: (Y, d) \rightarrow (E, \epsilon)$  with  $h \circ e' = e$ .

A *minimization* of a  $B$ -coalgebra  $(X, c)$  is a  $B$ -coalgebra  $(E, \epsilon)$  with a coalgebra morphism  $e: (X, c) \rightarrow (E, \epsilon)$  with  $e \in \mathcal{E}$  such that for every coalgebra morphism  $e': (X, c) \rightarrow (Y, d)$  with  $e' \in \mathcal{E}$  there is a unique coalgebra morphism  $h: (Y, d) \rightarrow (E, \epsilon)$  with  $h \circ e' = e$ . If a minimization exists then it is unique up to isomorphism, therefore we often speak about *the* minimization. If  $B$  has a final coalgebra  $(Z, \zeta)$  and  $B$  preserves  $\mathcal{M}$ -morphisms, then the minimization of  $(X, c)$  is equivalently given by  $(\mathcal{E}, \mathcal{M})$ -factorization (in  $\mathbf{coalg}(B)$ ) of the unique coalgebra morphism to  $(Z, \zeta)$ :

$$\begin{array}{ccccc} X & \xrightarrow{e} & E & \xrightarrow{m} & Z \\ c \downarrow & & \downarrow \epsilon & & \downarrow \zeta \\ BX & \xrightarrow{Be} & BE & \xrightarrow{Bm} & BZ \end{array}$$

The procedure from [1] for computing a minimization is based on the final sequence. We denote the poset category of ordinal numbers by **Ord**.

**Definition 4.2** The *final sequence*  $W: \text{Ord}^{\text{op}} \rightarrow \mathbf{C}$  of  $B$  is the unique sequence defined by  $W_0 = 1$  (the final object of  $\mathbf{C}$ ),  $W_{i+1} = BW_i$  and  $W_j = \lim_{i < j} W_i$  for a limit ordinal  $j$ , whose connecting morphisms  $w_{j,i}: W_j \rightarrow W_i$  (with  $i \leq j$ ) satisfy  $w_{i,i} = \text{id}$ ,  $w_{j+1,i+1} = Bw_{j,i}$  and if  $j$  is a limit ordinal then  $(w_{j,i})_{i < j}$  is a limit cone.

Any coalgebra  $c: X \rightarrow BX$  defines a unique cone  $(c_i: X \rightarrow W_i)_{i \in \text{Ord}}$  satisfying  $c_{i+1} = Bc_i \circ c$ . We use the notation  $c_i$  throughout this paper to refer to elements of the above cone, for a coalgebra  $(X, c)$ .

**Definition 4.3** For any coalgebra  $c: X \rightarrow BX$  and ordinal  $i$ , we define the *i-minimization* to be the  $\mathcal{E}$ -morphism  $e_i: X \rightarrow E_i$  of an  $(\mathcal{E}, \mathcal{M})$ -factorization of  $c_i$ .

The  $E_i$ 's form an ordinal indexed chain, with connecting morphisms  $e_{j,i}: E_j \rightarrow E_i$  (for  $i \leq j$ ) arising by diagonalization (so that  $e_i = e_{i+1,i} \circ e_{i+1}$  for all  $i$ ).

The following theorem collects what we need to know about  $(i)$ -minimizations. The first two items concern the existence of minimizations, and the third is a technique for computing  $i$ -minimizations.

**Theorem 4.4** [1] *Let  $c: X \rightarrow BX$  be a coalgebra.*

- (i) *Suppose that  $\mathcal{E}$  consists of epimorphisms, and suppose that the  $i$ -minimization  $e_i: X \rightarrow E_i$  of  $(X, c)$  is a coalgebra morphism from  $(X, c)$  to a  $B$ -coalgebra  $(E_i, \epsilon)$ . Then  $(E_i, \epsilon)$  is the minimization of  $(X, c)$ .*
- (ii) *In addition to the above assumptions, suppose  $\mathbf{C}$  is cowellpowered, and  $B$  preserves morphisms in  $\mathcal{M}$ . Then the minimization of any  $B$ -coalgebra exists, with carrier  $E_i$  for some ordinal number  $i$ .*
- (iii) *Suppose  $B$  preserves morphisms in  $\mathcal{M}$ , and  $e_i: X \rightarrow E_i$  is the  $i$ -minimization of  $(X, c)$ . Then the  $\mathcal{E}$ -morphism of an  $(\mathcal{E}, \mathcal{M})$ -factorization of  $Be_i \circ c: X \rightarrow BE_i$  is the  $(i+1)$ -minimization of  $(X, c)$ .*

**Example 4.5** Consider the **Set** functor  $BX = 2 \times X^A$ , whose coalgebras are deterministic automata, with the factorization system given by epis and monos. For an ordinal  $i$ , the set  $W_i$  in the final sequence of  $B$  consists of all languages over  $A$  where all words have length below  $i$ . Given a  $B$ -coalgebra  $(X, c)$ , the function  $c_i: X \rightarrow W_i$  maps a state  $x$  to the set of words of length below  $i$  accepted by  $x$ . Its kernel is the relation  $\equiv_i$  given in Section 2.1. Thus  $E_i$  is the quotient of states by  $\equiv_i$ . The inductive computation of  $e_i$  in Theorem 4.4(iii) underlies partition refinement algorithms for deterministic automata. For details and more examples, see [1].

## 5 Reachability

We define the notion of reachable part of an algebra, and a procedure to compute it. The definitions and results are dual to those of the previous section, but since they play an important role in the remainder of this paper we spell out some of the details, and state the dual of Theorem 4.4. Throughout this section, let  $\mathbf{D}$  be a cocomplete category with an  $(\mathcal{E}, \mathcal{M})$ -factorization system and  $L: \mathbf{D} \rightarrow \mathbf{D}$  a functor.

The *reachable part* of an  $L$ -algebra  $(X, a)$  is an  $L$ -algebra  $(R, \varrho)$  with a morphism  $m: (R, \varrho) \rightarrow (X, a)$  with  $m \in \mathcal{M}$ , satisfying the expected property dual to that of a minimization. If  $L$  has an initial algebra  $(A, \alpha)$  and  $L$  preserves  $\mathcal{E}$ -morphisms, then

the reachable part of  $(X, a)$  is equivalently given by  $(\mathcal{E}, \mathcal{M})$ -factorization (in  $\mathbf{alg}(L)$ ) of the unique algebra morphism from  $(A, \alpha)$  to  $(X, a)$ .

The *initial sequence*  $V: \mathbf{Ord} \rightarrow \mathbf{D}$  of  $L$  is the unique sequence defined by  $V_0 = 0$  (the initial object of  $\mathbf{D}$ ),  $V_{i+1} = LV_i$  and  $V_j = \text{colim}_{i < j} V_i$  for a limit ordinal  $j$ , whose connecting morphisms  $v_{i,j}: V_i \rightarrow V_j$  (with  $i \leq j$ ) satisfy  $v_{i,i} = \text{id}$ ,  $v_{i+1,j+1} = Lv_{i,j}$  and if  $j$  is a limit ordinal then  $(v_{i,j})_{i < j}$  is a colimit cocone.

Any algebra  $a: LX \rightarrow X$  defines a unique cocone  $(a_i: V_i \rightarrow X)_{i \in \mathbf{Ord}}$  satisfying  $a_{i+1} = a \circ La_i$ . We define the *i-reachable part* to be the  $\mathcal{M}$ -morphism  $m_i: R_i \rightarrow X$  of an  $(\mathcal{E}, \mathcal{M})$ -factorization of  $a_i$ . The  $R_i$ 's form an ordinal indexed chain, with connecting morphisms  $r_{i,j}: R_i \rightarrow R_j$  (for  $i \leq j$ ) arising by diagonalization (so that  $m_i = m_{i+1} \circ r_{i,i+1}$  for all  $i$ ).

**Theorem 5.1** *Let  $a: LX \rightarrow X$  be an algebra.*

- (i) *Suppose that  $\mathcal{M}$  consists of monomorphisms, and suppose that the i-reachable part  $m_i: R_i \rightarrow X$  of  $(X, a)$  is an algebra morphism from an L-algebra  $(R_i, \varrho)$  to  $(X, a)$ . Then  $(R_i, \varrho)$  is the reachable part of  $(X, a)$ .*
- (ii) *In addition to the above assumptions, suppose  $\mathbf{D}$  is wellpowered, and  $L$  preserves morphisms in  $\mathcal{E}$ . Then the reachable part of any L-algebra exists, with carrier  $R_i$  for some ordinal number  $i$ .*
- (iii) *Suppose  $L$  preserves morphisms in  $\mathcal{E}$ , and  $m_i: R_i \rightarrow X$  is the i-reachable part of  $(X, a)$ . Then the  $\mathcal{M}$ -morphism of an  $(\mathcal{E}, \mathcal{M})$ -factorization of  $a \circ Lm_i: LR_i \rightarrow X$  is the  $(i+1)$ -reachable part of  $(X, a)$ .*

**Example 5.2** Let  $L$  be the **Set** endofunctor defined by  $LX = A \times X + 1$ . As explained in Section 3, an algebra  $[g, \iota]: A \times X + 1 \rightarrow X$  is a deterministic automaton with initial state  $\iota(*)$ , transition function  $g$  and no final states. A set  $V_i$  in the initial sequence of  $L$  is the set of words of length below  $i$ , and the function  $[g, \iota]_i: V_i \rightarrow X$  maps  $w \in V_i$  to the state that is reached after processing  $w$  from right to left:  $[g, \iota]_i(\varepsilon) = \iota(*)$  and, for  $a \in A$  and  $w \in V_{i-1}$ ,  $[g, \iota]_i(aw) = g(a, [g, \iota]_i(w))$ .

The *i-reachable part*  $m_i: R_i \rightarrow X$  is concretely presented by letting  $R_i$  be the set of states reachable from words of length below  $i$ , and  $m_i$  the inclusion map. For  $i = 0$  we have  $V_0 = \emptyset$ , hence  $R_0 = \emptyset$ . The computation of  $m_{i+1}: R_{i+1} \rightarrow X$  from  $m_i$  in Theorem 5.1 amounts to taking the image of  $LR_i$  along  $[g, \iota] \circ Lm_i$ , i.e.,  $R_{i+1} = \{g(a, m_i(x)) \mid a \in A, x \in R_i\} \cup \{\iota(*)\}$ . The reachable part of  $(X, [g, \iota])$  consists of all states that are reachable from some word in  $A^*$ , starting from the initial state.

## 6 Minimization via reachability

We formulate the minimization construction sketched in Section 2.2 in terms of (co)algebras. The instantiation to deterministic automata is presented in Example 6.2, which can be read without necessarily understanding the abstract construction. For the abstract construction, we assume:

- (A1) categories  $\mathbf{C}$  and  $\mathbf{D}$ , both with (epi,mono)-factorization systems;
- (A2) a functor  $B: \mathbf{C} \rightarrow \mathbf{C}$  that preserves epis;

- (A3) a functor  $L: \mathbf{D} \rightarrow \mathbf{D}$  that preserves monos;
- (A4) a (contravariant) adjunction between functors  $F: \mathbf{C}^{\text{op}} \rightarrow \mathbf{D}$  and  $G: \mathbf{D}^{\text{op}} \rightarrow \mathbf{C}$ ;
- (A5) a natural isomorphism  $\rho: BG \Rightarrow GL$ ;
- (A6) the existence of an initial  $L$ -algebra.

By (A1) ... (A3), both  $\mathbf{C}$  and  $\mathbf{D}$  have (epi,mono)-factorization systems that extend to  $\text{coalg}(B)$  and  $\text{alg}(L)$  respectively. The contravariant adjunction of (A4) lifts, using the isomorphism in (A5), to a (contravariant) adjunction between  $\bar{F}: \text{coalg}(B)^{\text{op}} \rightarrow \text{alg}(L)$  and  $\bar{G}: \text{alg}(L)^{\text{op}} \rightarrow \text{coalg}(B)$  (see [10], and also [13,15]).

**Theorem 6.1** *Assume (A1) ... (A6) from the beginning of this section, and let  $(X, c)$  be a  $B$ -coalgebra. Let  $m: (R, \varrho) \rightarrow \bar{F}(X, c)$  be the reachable part of  $\bar{F}(X, c)$ . Take an (epi,mono)-factorization (in  $\text{coalg}(B)$ ) of the adjoint transpose  $m^\flat$  of  $m$ :*

$$(X, c) \xrightarrow{\quad m^\flat \quad} (E, \epsilon) \xrightarrow{\quad \quad \quad} \bar{G}(R, \varrho) \quad (3)$$

Then  $(E, \epsilon)$  is the minimization of  $(X, c)$ .

The functor  $\bar{F}$  is defined on objects by  $\bar{F}(X, c) = (FX, Fc \circ \rho_X^\flat)$ , where  $\rho^\flat: LF \Rightarrow FB$  is the mate of  $\rho$ , and  $\bar{G}$  by  $\bar{G}(X, a) = (GX, \rho_X^{-1} \circ Ga)$ . See [14,15] for details. We often abbreviate  $Fc \circ \rho_X^\flat$  by  $\bar{F}c$ , and in particular we write  $((\bar{F}c)_i: V_i \rightarrow FX)_{i \in \text{Ord}}$  for the cocone over the initial sequence of  $L$  induced by  $\bar{F}(X, c)$ .

The construction in Theorem 6.1 is based on [15], which in turn is based on techniques from coalgebraic modal logic. Indeed, a natural transformation  $\rho$  of the above form (without the assumption that it is an iso) is by now a standard way of defining the semantics of coalgebraic modal logic, see, e.g., [14,17].

The minimization construction of [15] concerns a more general class of coalgebras, that may involve branching. As explained in Section 8, the factorization of  $m^\flat$  yielding a minimal automaton can not be formulated in that setting. The construction is also connected to the one in [4], which however assumes a duality rather than a contravariant adjunction (making the factorization of  $m^\flat$  unnecessary, since it is automatically an epi because of the duality). That construction rules out our example of deterministic automata below.

**Example 6.2** We apply the construction of Theorem 6.1 to deterministic automata over an alphabet  $A$ . The ingredients (A1) ... (A6) of the beginning of this section are as follows:  $\mathbf{C} = \mathbf{D} = \mathbf{Set}$ ,  $F = G = 2^-$  (the contravariant powerset functor),  $B: \mathbf{Set} \rightarrow \mathbf{Set}$  is given by  $BX = 2 \times X^A$ ,  $L: \mathbf{Set} \rightarrow \mathbf{Set}$  is given by  $LX = A \times X + 1$ . The required isomorphism  $\rho: BG \Rightarrow GL$  is  $2 \times (2^-)^A \cong 2^{A \times - + 1}$ . Recall that  $L$  has an initial algebra, given by the set of words  $A^*$ .

Let  $\langle o, f \rangle: X \rightarrow 2 \times X^A$  be a  $B$ -coalgebra. The first step of the construction is to compute  $\bar{F}(X, \langle o, f \rangle) = (2^X, 2^{\langle o, f \rangle} \circ \rho_X^\flat)$ , which we denote by  $[g, \iota]: A \times 2^X + 1 \rightarrow 2^X$ . Intuitively,  $(2^X, [g, \iota])$  is obtained by reversing and determinizing the automaton  $(X, \langle o, f \rangle)$ , where reversal comes from the application  $2^{\langle o, f \rangle}$  of the contravariant powerset functor. By computing the mate  $\rho^\flat$  of  $\rho$ , we obtain (see [15,23] for details):

$$g(a, S) = \{x \in X \mid f(x)(a) \in S\} \quad \text{and} \quad \iota(*) = \{x \in X \mid o(x) = 1\}.$$

The reachable part  $R \subseteq 2^X$  (technically, an inclusion map  $m: R \rightarrow 2^X$ ) consists of all reachable (sets of) states in  $(2^X, [g, \iota])$ . By Theorem 5.1,  $R$  can be obtained by computing  $i$ -reachable parts by induction on  $i$ , according to (see Example 5.2):

$$R_{i+1} = \{\{x \in X \mid f(x)(a) \in S\} \mid a \in A, S \in R_i\} \cup \{\{x \in X \mid o(x) = 1\}\}$$

and  $R_0 = \emptyset$ . We thus retrieve the reachable sets as constructed in Section 2.2.

Following Theorem 6.1, we compute an (epi,mono)-factorization of the transpose  $m^b$  of  $m$ , and obtain a coalgebra  $(E, \epsilon)$  which is the minimization of  $(X, \langle o, f \rangle)$ . The transpose  $m^b: X \rightarrow 2^R$  is given by  $m^b(x) = \{S \in R \mid x \in S\}$ . Concretely, the factorization  $E$  can be defined as the image of  $X$  along  $m^b$ . But observe that we can also define  $e: X \rightarrow E$  (and, implicitly,  $E$ ) by  $e(x) = \{y \mid \forall S \in R: x \in S \text{ iff } y \in S\}$ . Then  $E$  is the quotient of  $X$  by language equivalence, see Section 2.2.

## 7 Relating minimization and reachability

We have seen how minimization can be computed either by a stepwise computation along the final sequence, or by a stepwise computation along an initial sequence followed by a factorization. Next we show that, when both approaches apply, there is a strong correspondence: the arrows from the initial sequence and those into the final sequences are each others adjoint transpose, up to isomorphism (Theorem 7.2). Based on this correspondence, we derive an abstract method to compute the  $i$ -th partition from the  $i$ -th reachability step (Corollary 7.3), generalizing the computation of  $\equiv_i$  (or  $E_i$ ) from  $R_i$  in Section 2.2.

Throughout this section we assume (A1) . . . (A5) from the beginning of Section 6, i.e., categories  $\mathbf{C}$  and  $\mathbf{D}$  with (epi,mono)-factorization systems, functors  $B: \mathbf{C} \rightarrow \mathbf{C}$  preserving monos and  $L: \mathbf{D} \rightarrow \mathbf{D}$  preserving epis, a contravariant adjunction between  $F$  and  $G$  and finally a natural iso  $\rho: BG \Rightarrow GL$ . We further assume that  $\mathbf{C}$  is complete and  $\mathbf{D}$  is cocomplete.

**Lemma 7.1** *Let  $W: \text{Ord}^{\text{op}} \rightarrow \mathbf{C}$  be the final sequence of  $B$ , and  $V: \text{Ord} \rightarrow \mathbf{D}$  the initial sequence of  $L$ . There is a natural isomorphism  $\kappa: W \Rightarrow GV^{\text{op}}: \text{Ord}^{\text{op}} \rightarrow \mathbf{C}$  satisfying  $\kappa_{i+1} = \rho_{V_i} \circ B\kappa_i$  for all ordinals  $i$ .*

The following is the heart of the matter, relating the cone  $(c_i: X \rightarrow W_i)_{i \in \text{Ord}}$  over the final sequence of  $B$  induced by  $(X, c)$  to the cocone  $((\overline{F}c)_i: V_i \rightarrow FX)_{i \in \text{Ord}}$  over the initial sequence of  $L$  induced by  $\overline{F}(X, c)$ .

**Theorem 7.2** *Let  $(X, c)$  be a  $B$ -coalgebra. For any ordinal  $i$ , the following diagram commutes:*

$$\begin{array}{ccc} X & \xrightarrow{c_i} & W_i \\ & \searrow (\overline{F}c)_i^b & \downarrow \kappa_i \\ & & GV_i \end{array}$$

**Corollary 7.3** *Let  $(X, c)$  be a  $B$ -coalgebra. Let  $m_i^b: X \rightarrow GR_i$  be the transpose of the  $i$ -reachable part of  $\overline{F}(X, c)$ . Then the epic morphism  $e_i: X \rightarrow E_i$  of an (epi,mono)-factorization of  $m_i^b$  is the  $i$ -minimization of  $(X, c)$ . Further, if  $m_i: R_i \rightarrow FX$  is the reachable part of  $\overline{F}(X, c)$ , then  $e_i$  is the minimization of  $(X, c)$ .*

**Example 7.4** In Section 2.2 we have seen how the  $i$ -th partition of the states of a deterministic automaton can be obtained from the sets of states reachable in the reversed determinized automaton in less than  $i$  steps, by interpreting the reachable sets as splitters. This result is a special case (and, indeed, we derived it from) Corollary 7.3. To see this, let  $(X, c)$  be a deterministic automaton, recall from Example 4.5 that the  $i$ -th partition is the  $i$ -minimization of  $(X, c)$ , and recall from Example 6.2 that the sets of states reachable in the reversed determinized automaton are given by the  $i$ -reachable part  $m_i$  of  $\overline{F}(X, c)$ . The “splitting” operation corresponds to a specific factorization of  $m_i^b$ , similar to the last part of Example 6.2.

One may wonder whether there is a converse, i.e., if we can obtain the  $i$ -reachable part of  $\overline{F}(X, c)$  from the  $i$ -minimization of  $(X, c)$ . Example 2.1 shows that this is not the case: partition refinement for deterministic automata may stop earlier than the computation of reachable sets in the reversed determinized automaton.

Under the assumptions of Theorem 5.1, the reachable part of any  $L$ -algebra arises as one of the  $i$ -reachable parts, hence Corollary 7.3 shows that, in that case, Theorem 6.1 holds even if  $L$  does not have an initial algebra.

We can also use Corollary 7.3 to *combine* the minimization procedure based on  $i$ -minimizations and the one based on  $i$ -reachable parts. The possibility of computing the  $i$ -minimization from the  $i$ -reachable part suggests a procedure where we inductively compute  $i$ -reachable parts as in Theorem 5.1, compute  $i$ -minimizations along the way and terminate when the  $i$ -minimization is a minimization.

In this procedure, when computing the  $(i + 1)$ -minimization from the  $(i + 1)$ -reachable part, one would like to use the  $i$ -minimization as well. Concretely, for deterministic automata, given the partition  $E_i$  computed from the splitters  $R_i$  (Section 2.2), and the new set of splitters  $R_{i+1}$ , we want to compute  $E_{i+1}$  by splitting the partition in  $E_i$  according to the new splitters, i.e., those appearing in  $R_{i+1}$  but not in  $R_i$ . Abstractly, one can compute  $E_{i+1}$  from  $E_i$ ,  $R_i$  and  $R_{i+1}$  as follows.

**Lemma 7.5** *Suppose  $\mathbf{C}$  has pullbacks. Let  $e_i: X \rightarrow E_i$  be the  $i$ -minimization of a coalgebra  $c: X \rightarrow BX$ , and let  $r_{i,i+1}: R_i \rightarrow R_{i+1}$  be the arrow (see Section 5) from the  $i$ -reachable part  $m_i: R_i \rightarrow FX$  to the  $(i + 1)$ -reachable part  $m_{i+1}: R_{i+1} \rightarrow FX$  of  $\overline{F}(X, c)$ . By Corollary 7.3,  $m_i^b = m' \circ e_i$  for some mono  $m'$ . Let  $P$  be the pullback of  $m'$  and  $Gr_{i,i+1}$ :*

$$\begin{array}{ccc}
 X & \xrightarrow{m_{i+1}^b} & GR_{i+1} \\
 \searrow h & \downarrow \lrcorner & \downarrow Gr_{i,i+1} \\
 P & \xrightarrow{\quad} & GR_{i+1} \\
 \downarrow e_i & & \downarrow \\
 E_i & \xrightarrow{m'} & GR_i
 \end{array}$$

*There is a unique mediating morphism  $h$  as above. The epic part of an  $(\text{epi}, \text{mono})$ -factorization of  $h$  is the  $(i + 1)$ -minimization of  $(X, c)$ .*

To understand the above construction, consider the case of deterministic automata, with  $E_i$  presented as a partition and  $R_i$  as a set of splitters, as above. The pullback  $P$  can be presented by  $P = \{(Q, C) \in E_i \times 2^{R_{i+1}} \mid C \subseteq R_{i+1} \setminus R_i\}$  (see the appendix). The function  $h: X \rightarrow P$  maps  $x$  to the pair  $(e_i(x), \{S \in R_{i+1} \setminus R_i \mid x \in$



$X\}$ ), i.e., the pair consisting of the equivalence class of  $x$  in  $E_i$  and the set of all “new” splitters, appearing in  $R_{i+1}$  but not in  $R_i$ , containing  $x$ . The factorization of  $h$  can be presented by mapping each  $x$  to  $\{y \in e_i(x) \mid \forall S \in R_{i+1} \setminus R_i : x \in S \text{ iff } y \in S\}$ , yielding the partition obtained by splitting  $E_i$  according to all the new splitters.

For deterministic automata, the inductive computation of  $i$ -reachable parts, and  $i$ -minimizations from them using Corollary 7.3 and Lemma 7.5 closely resembles the construction presented in [8, Algorithm 1] and the end of Section 2.2. However, the algorithm in [8] terminates only when the reachable part  $R$  has been found, whereas using Corollary 7.3 we can terminate once the  $i$ -minimization is a minimization. This may occur before the  $i$ -reachable part is the reachable part (Example 2.1).

## 8 Branching systems

In the previous sections, we studied minimization of  $B$ -coalgebras, with deterministic automata as the main example. Next, we investigate the case of systems involving branching, such as non-deterministic or alternating automata. Here, we do not focus on finding minimal non-deterministic automata: it is well-known that they are not unique, and it is in fact much less obvious how to even define the notion of minimization. Instead, we show how to compute language equivalence inductively based on reachability.

*Language semantics.* We are interested in coalgebras for a composite functor of the form  $BT$  or  $TB$ , where  $B$  models the observations that are to be recorded in traces, and  $T$  is the type of branching. For instance, taking  $BX = 2 \times X^A$  as before and  $T = \mathcal{P}$  the (covariant) powerset functor,  $BT$ -coalgebras are non-deterministic automata; and with  $T = \mathcal{PP}$ ,  $BT$ -coalgebras are a form of alternating automata. Taking  $B$  to be a polynomial functor and  $T = \mathcal{P}$  one obtains tree automata as  $TB$ -coalgebras, and for a certain choice of  $T$  one obtains weighted tree automata [15]. Because of space limitations, we focus on  $BT$ -coalgebras in this section, and only treat the example of non-deterministic automata.

The final semantics of  $BT$ -coalgebras such as those in the above examples (which exists, for instance, when we restrict  $T$  to the finite powerset functor) does, in general, not coincide with the expected language semantics. We recall the approach of [15] to define language semantics based on initial algebras rather than final coalgebras. To this end, assume functors  $B, T: \mathbf{C} \rightarrow \mathbf{C}$ , a functor  $L: \mathbf{D} \rightarrow \mathbf{D}$  with an initial algebra and, as before (Section 6), a contravariant adjunction between  $F$  and  $G$ . To define language semantics, we assume a natural transformation  $\rho: BG \Rightarrow GL$  (not necessarily an isomorphism) and a natural transformation  $\alpha: TG \Rightarrow G$ . This induces a functor  $\bar{F}_\alpha: \text{coalg}(BT) \rightarrow \text{alg}(L)$  defined by  $\bar{F}_\alpha(X, c) = (FX, Fc \circ \rho_{TX}^b \circ L\alpha^b)$ , see [15] for details and explanation. Given a coalgebra  $c: X \rightarrow BTX$ , one then computes the unique map  $s: (A, \alpha) \rightarrow \bar{F}_\alpha(X, c)$  from the initial  $L$ -algebra, and defines the *(language) semantics* of  $(X, c)$  to be the transpose  $s^b: X \rightarrow GA$  of  $s$ . We define the *language quotient* of  $(X, c)$  as the epic part of an (epi,mono)-factorization of the language semantics  $s^b$ .

**Theorem 8.1** *Suppose that  $\mathbf{C}$  and  $\mathbf{D}$  have (epi,mono)-factorization systems. Let  $c: X \rightarrow BTX$  be a coalgebra, and let  $m: (R, \varrho) \rightarrow \bar{F}_\alpha(X, c)$  be the reachable part of*

$\overline{F}_\alpha(X, c)$ . Then the epic part of an (epi,mono)-factorization (in  $\mathbf{C}$ ) of the transpose  $m^b: X \rightarrow GR$  is the language quotient of  $(X, c)$ .

**Example 8.2** Let  $F = G = 2^-$  be the contravariant powerset adjunction, let  $BX = 2 \times X^A$  and  $LX = A \times X + 1$ . A non-deterministic automaton is a coalgebra  $\langle o, f \rangle: X \rightarrow 2 \times (\mathcal{P}X)^A$  for the composite functor  $B\mathcal{P}$ . Define the components of  $\alpha: \mathcal{P}2^- \Rightarrow 2^-$  by union, and let  $\rho$  be the isomorphism from Example 6.2.

We denote the algebra  $\overline{F}_\alpha(X, \langle o, f \rangle)$  by  $[g, \iota]: A \times 2^X + 1 \rightarrow 2^X$ . It is given by

$$g(a, S) = \{x \mid \exists y \in f(x)(a) \text{ s.t. } y \in S\} \quad \iota(*) = \{x \mid o(x) = 1\}$$

(cf. Example 6.2). Hence, the unique algebra morphism  $s: A^* \rightarrow 2^X$  satisfies  $s(\varepsilon) = \{x \mid o(x) = 1\}$  and  $s(aw) = \{x \mid \exists y \in f(x)(a) \text{ s.t. } y \in s(w)\}$ . The transpose  $s^b$  is thus the usual semantics of non-deterministic automata [15].

The reachable part  $R \subseteq 2^X$  consists of all reachable (sets of) states in  $(2^X, [g, \iota])$ . By Theorem 5.1,  $R$  can be obtained by computing  $i$ -reachable parts by induction on  $i$ , according to (see Example 5.2)  $R_0 = \emptyset$  and:

$$R_{i+1} = \{\{x \mid \exists y \in f(x)(a) \text{ s.t. } y \in S\} \mid a \in A, S \in R_i\} \cup \{\{x \in X \mid o(x) = 1\}\}.$$

The function  $m^b: X \rightarrow 2^R$  maps every state  $x$  to those sets in  $R$  that contain it, and, like in Example 6.2, we may define the epic part of a factorization of  $m^b$  by  $e(x) = \{y \in X \mid \forall S \in R: x \in S \text{ iff } y \in S\}$ . Then  $e$  maps every state  $x$  to the equivalence class of states that accept the same language.

It was shown in [15] that, in the context of Theorem 8.1, if the natural transformation  $\rho$  is an isomorphism, then  $GR$  is a  $B$ -coalgebra, whose unique morphism  $h$  to the final coalgebra is mono, and such that  $s^b = h \circ m^b$ . This means that the construction yields a  $B$ -coalgebra whose states are behaviourally equivalent if and only if they are equal, and whose final semantics represents the language semantics of the original automaton. Instances where the conditions of the construction are met include non-deterministic, alternating and weighted automata, see [15]. Here, our characterization of reachable sets shows how to compute the factorization of the morphism from the initial algebra.

The construction from [15] mentioned above is reminiscent of Brzozowski's minimization algorithm, but it does not generalize the construction for  $B$ -coalgebras in Theorem 6.1. The latter is based on another (epi,mono)-factorization in  $\mathbf{coalg}(B)$ . In the example of non-deterministic automata, the construction from [15] yields a deterministic automaton, which is not minimal in any reasonable sense: it may contain states that are not reachable from any state in the image of  $X$  along  $m^b$ . Note that the reachable states can not be obtained in general by taking the image of the state space  $X$  along  $m^b$ , since the minimal deterministic automaton may have more states than the non-deterministic one that we start with. Instead, one should construct the *least subautomaton* containing this image. In  $\mathbf{Set}$ , this is easy to define (e.g., [21]), but at the abstract level it seems less clear.

*Language equivalence: a dual view.* We briefly consider a construction for branching systems that is not unlike the minimization construction of Section 4. To this end, suppose  $\mathbf{C}$  is complete and  $\mathbf{D}$  is cocomplete, and let  $V$  be the initial sequence of



$L: D \rightarrow D$ . Given  $c: X \rightarrow BTX$ , there is a unique cone  $(\bar{c}_i: X \rightarrow GV_i)_{i \in \text{Ord}}$  over  $GV^{\text{op}}$  satisfying the following:

$$\bar{c}_{i+1} = (X \xrightarrow{c} BTX \xrightarrow{BT\bar{c}_i} BTGV_i \xrightarrow{B\alpha_{V_i}} BGV_i \xrightarrow{\rho_{V_i}} GLV_i).$$

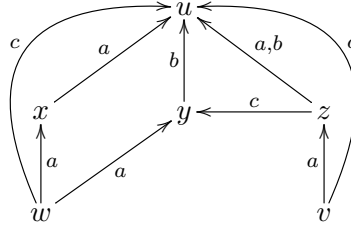
Call the epic morphism of an (epi,mono)-factorization of  $\bar{c}_i$  the *i-language quotient* of  $(X, c)$ . (If  $\rho: BG \Rightarrow GL$  is an isomorphism, then the above cone can equivalently be defined over the final sequence.)

**Theorem 8.3** *Let  $c: X \rightarrow BTX$  be a coalgebra, and  $((\bar{F}_\alpha c)_i: V_i \rightarrow FX)_{i \in \text{Ord}}$  the cocone over the initial sequence of  $L$  induced by  $\bar{F}_\alpha(X, c)$ . For any  $i$ , we have  $\bar{c}_i = (\bar{F}_\alpha c)_i^b$ . Further, let  $m_i: R_i \rightarrow FX$  be the  $i$ -reachable part of  $\bar{F}_\alpha(X, c)$ . Then the epic morphism of an (epi,mono)-factorization of the transpose  $m_i^b: X \rightarrow GR_i$  is the  $i$ -language quotient of  $(X, c)$ .*

The crucial property for the minimization construction in Theorem 4.4 is that the  $(i+1)$ -minimization can be computed from the  $i$ -minimization. This approach does not seem to work for  $i$ -language quotients, since  $\alpha$  and  $\rho$  are not (componentwise) mono in general. Indeed, for non-deterministic automata,  $E_i$  is the quotient of states by language equivalence of words with length below  $i$ , and it is unclear how one could obtain  $E_{i+1}$  only from  $E_i$  and the automaton under consideration.

In the previous section (Lemma 7.5), we have seen how the  $(i+1)$ -minimization can be obtained given the  $(i+1)$ -reachable part and the  $i$ -minimization. A similar approach could be taken here, generating a sequence of  $i$ -language quotients. However, it is not clear whether this is of much use. The problem is that, in the current context, it may be the case that the  $i$ -language quotient is isomorphic to the  $(i+1)$ -language quotient, but not to the  $j$ -language quotient for some  $j > i+1$ . Hence, we can not use such an isomorphism as a termination condition.

**Example 8.4** Consider the following non-deterministic automaton, where the only accepting state is  $u$ .



The  $i$ -reachable sets  $R_i$ , as computed in Example 8.2, and the  $i$ -language quotients  $E_i$ , which we compute from the  $R_i$ 's (Theorem 8.3), are:

$$\begin{aligned} E_0 &= \{\{u, x, y, z, w, v\}\} & R_0 &= \emptyset \\ E_1 &= \{\{u\}, \{x, y, z, w, v\}\} & R_1 &= \{\{u\}\} \\ E_2 &= \{\{u\}, \{w, v\}, \{z\}, \{x\}, \{y\}\} & R_2 &= \{\{u\}, \{w, v\}, \{x, z\}, \{y, z\}\} \\ E_3 &= \{\{u\}, \{w, v\}, \{z\}, \{x\}, \{y\}\} & R_3 &= \{\{u\}, \{w, v\}, \{x, z\}, \{y, z\}, \emptyset, \{z\}\} \\ E_4 &= \{\{u\}, \{w\}, \{v\}, \{z\}, \{x\}, \{y\}\} & R_4 &= \{\{u\}, \{w, v\}, \{x, z\}, \{y, z\}, \emptyset, \{z\}, \{v\}\} \end{aligned}$$

Notice that  $E_3 = E_2$ , but  $E_4 \neq E_3$ . Indeed, all states except  $w$  and  $v$  are distinguished by the empty word or a word of length 1, whereas it requires a word of length 3 to distinguish  $w$  and  $v$ .

## 9 Future work

We established a connection between partition refinement and Brzozowski’s minimization construction, based on an abstract coalgebraic perspective. Our interest was to understand deterministic automata, which is hence the one example we cover in detail. The necessary assumptions of our results are also satisfied by Moore automata (and stream systems), and potentially other examples (e.g., [23]). In particular, it would be interesting to use the dualities of [20] and our results on branching systems to develop generic constructions for canonical branching systems. In this context, the connection to weak factorization systems as used in [1] and the approach of [16] also remain to be understood. Further, the interaction between minimization and coalgebraic determinization constructions is left open.

## References

- [1] Adámek, J., F. Bonchi, M. Hülsbusch, B. König, S. Milius and A. Silva, *A coalgebraic perspective on minimization and determinization*, in: *Procs. FOSSACS 2012*, LNCS **7213**, 2012, pp. 58–73.
- [2] Adámek, J., H. Herrlich and G. E. Strecker, “Abstract and Concrete Categories - The Joy of Cats,” Dover Publications, 2009.
- [3] Berstel, J., L. Boasson, O. Carton and I. Fagnot, *Minimization of automata*, CoRR **abs/1010.5318** (2010), to appear in the Handbook of Automata.
- [4] Bezhanishvili, N., C. Kupke and P. Panangaden, *Minimization via duality*, in: *Procs. WoLLIC 2012*, LNCS **7456**, 2012, pp. 191–205.
- [5] Bonchi, F., M. M. Bonsangue, H. H. Hansen, P. Panangaden, J. J. M. M. Rutten and A. Silva, *Algebra-coalgebra duality in Brzozowski’s minimization algorithm*, ACM Trans. Comput. Log. **15** (2014), p. 3.
- [6] Bonchi, F., M. M. Bonsangue, J. J. M. M. Rutten and A. Silva, *Brzozowski’s algorithm (co)algebraically*, in: *Logic and Program Semantics - Essays Dedicated to Dexter Kozen on the Occasion of His 60th Birthday*, 2012, pp. 12–23.
- [7] Brzozowski, J., *Canonical regular expressions and minimal state graphs for definite events.*, Mathematical Theory of Automata **12** (1962), pp. 529–561.
- [8] Champarnaud, J., A. Khorsi and T. Paranthoën, *Split and join for minimizing: Brzozowski’s algorithm*, in: M. Balík and M. Simánek, editors, *Proceedings of the Prague Stringology Conference 2002, Prague, Czech Republic, September 23-24, 2002* (2002), pp. 96–104.
- [9] García, P., D. López and M. Vázquez de Parga, *DFA minimization: Double reversal versus split minimization algorithms*, Theor. Comput. Sci. **583** (2015), pp. 78–85.
- [10] Hermida, C. and B. Jacobs, *Structural induction and coinduction in a fibrational setting*, Inf. and Comp. **145** (1997), pp. 107–152.
- [11] Hopcroft, J. E., *An  $n \log n$  algorithm for minimizing states in a finite automaton*, in: *Theory of Machines and Computations*, 1971, pp. 189–196.
- [12] Hopcroft, J. E. and J. D. Ullman, “Introduction to Automata Theory, Languages and Computation,” Addison-Wesley, 1979.
- [13] Jacobs, B., A. Silva and A. Sokolova, *Trace semantics via determinization*, J. Comput. Syst. Sci. **81** (2015), pp. 859–879.
- [14] Klin, B., *Coalgebraic modal logic beyond sets*, ENTCS **173** (2007), pp. 177–201.
- [15] Klin, B. and J. Rot, *Coalgebraic trace semantics via forgetful logics*, in: *FoSSaCS 2015. Proceedings*, 2015, pp. 151–166.

- [16] König, B. and S. Küpper, *Generic partition refinement algorithms for coalgebras and an instantiation to weighted automata*, in: *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, 2014, pp. 311–325.
- [17] Kupke, C. and D. Pattinson, *Coalgebraic semantics of modal logics: An overview*, Theor. Comput. Sci. **412** (2011), pp. 5070–5094.
- [18] Kurz, A., “Logics for coalgebras and applications to computer science,” Ph.D. thesis, Ludwigs-Maximilians-Universität München (2000).
- [19] Moore, E. F., *Gedanken-experiments on sequential machines*, Automata studies **34** (1956), pp. 129–153.
- [20] Myers, R. S. R., J. Adámek, S. Milius and H. Urbat, *Coalgebraic constructions of canonical nondeterministic automata*, Theor. Comput. Sci. **604** (2015), pp. 81–101.
- [21] Rutten, J. J. M. M., *Universal coalgebra: a theory of systems*, Theor. Comput. Sci. **249** (2000), pp. 3–80.
- [22] Sakarovitch, J., “Elements of Automata Theory,” Cambridge University Press, 2009.
- [23] Salamanca, J., M. Bonsangue and J. Rot, *Duality of equations and coequations via contravariant adjunctions*, in: *CMCS*, 2016, to appear.
- [24] Watson, B. W., “Taxonomies and Toolkits of Regular Language Algorithms,” Ph.D. thesis, Eindhoven University of Technology (1995).

## A Proofs of Section 4

Theorem 4.4 is proved in [1] (with item (i) inlined in the proof of item (ii)). Because of this presentation difference, and for convenience, we recall the proof here. First, we need the following technicality, see [1].

**Lemma A.1** *Let  $h: (X, c) \rightarrow (Y, d)$  be a coalgebra homomorphism. Then  $c_i = d_i \circ h$  for all ordinals  $i$ .*

**Theorem 4.4** [1] *Let  $c: X \rightarrow BX$  be a coalgebra.*

- (i) *Suppose that  $\mathcal{E}$  consists of epimorphisms, and suppose that the  $i$ -minimization  $e_i: X \rightarrow E_i$  of  $(X, c)$  is a coalgebra morphism from  $(X, c)$  to a  $B$ -coalgebra  $(E_i, \epsilon)$ . Then  $(E_i, \epsilon)$  is the minimization of  $(X, c)$ .*
- (ii) *In addition to the above assumptions, suppose  $\mathbf{C}$  is cowellpowered, and  $B$  preserves morphisms in  $\mathcal{M}$ . Then the minimization of any  $B$ -coalgebra exists, with carrier  $E_i$  for some ordinal number  $i$ .*
- (iii) *Suppose  $B$  preserves morphisms in  $\mathcal{M}$ , and  $e_i: X \rightarrow E_i$  is the  $i$ -minimization of  $(X, c)$ . Then the  $\mathcal{E}$ -morphism of an  $(\mathcal{E}, \mathcal{M})$ -factorization of  $Be_i \circ c: X \rightarrow BE_i$  is the  $(i + 1)$ -minimization of  $(X, c)$ .*

**Proof.**

- (i) By assumption,  $e_i$  is a coalgebra homomorphism from  $(X, c)$  to  $(E_i, \epsilon)$ . Let  $h: (X, c) \rightarrow (Y, d)$  be a coalgebra morphism with  $h \in \mathcal{E}$ . By Lemma A.1, the upper right triangle in the diagram on the left-hand side commutes:

$$\begin{array}{ccc} X & \xrightarrow{h} & Y \\ e_i \downarrow & \searrow c_i & \downarrow d_i \\ E_i & \xrightarrow{m_i} & W_i \end{array} \qquad \begin{array}{ccc} X & \xrightarrow{h} & Y \\ e_i \downarrow & \searrow e' & \downarrow d_i \\ E_i & \xrightarrow{m_i} & W_i \end{array}$$

By Definition 4.3, the  $i$ -minimization of  $(X, c)$  is an  $(\mathcal{E}, \mathcal{M})$ -factorization of  $c_i$  with  $\mathcal{E}$ -morphism  $e_i$ ; we denote the  $\mathcal{M}$ -morphism by  $m_i$ , hence the lower left triangle commutes by definition. As a consequence of commutativity of the square, we obtain a unique diagonal  $e'$  making the diagram on the right-hand side commute. It only remains to be shown that  $e'$  is a coalgebra morphism. This follows since  $h$  is epic and both  $e' \circ h = e_i$  and  $h$  are coalgebra morphisms [21, Lemma 2.4].

- (ii) First, for any given  $i$ , let  $e_i: X \rightarrow E_i$  be the  $i$ -minimization of  $(X, c)$ , with corresponding  $\mathcal{M}$ -morphism  $m_i$  (i.e., such that  $c_i = m_i \circ e_i$ ). The outside of the following diagram commutes:

$$\begin{array}{ccc} X & \xrightarrow{e_{i+1}} & E_{i+1} \\ c \downarrow & \searrow \epsilon_i & \downarrow m_{i+1} \\ BX & \xrightarrow{\epsilon_i} & E_i \\ Be_i \downarrow & \searrow & \downarrow \\ BE_i & \xrightarrow{Bm_i} & BW_i \end{array}$$

where  $Bm_i$  is in  $\mathcal{M}$  by assumption. Thus, we obtain a diagonal  $\epsilon$ .

As explained in [1], since  $\mathbf{C}$  is cowellpowered and the  $e_i$ 's form a chain of epimorphisms with domain  $X$ , there is an  $i$  such that the arrow  $e_{i+1,i}: E_{i+1} \rightarrow E_i$  is an isomorphism. We denote its inverse by  $\iota: E_i \rightarrow E_{i+1}$ , then  $\iota \circ e_i = e_{i+1}$  (since  $e_i = e_{i+1,i} \circ e_{i+1}$ , see Section 4). We obtain a coalgebra on  $E_i$  turning  $e_i$  into a coalgebra morphism:

$$\begin{array}{ccc} X & \xrightarrow{e_i} & E_i \\ & \searrow e_{i+1} & \downarrow \iota \\ & & E_{i+1} \\ \downarrow c & & \downarrow \epsilon_i \\ BX & \xrightarrow{Be_i} & BE_i \end{array}$$

By (i), the coalgebra on  $E_i$  is the minimization of  $(X, c)$ .

- (iii) Let  $m_i$  be the  $\mathcal{M}$ -morphism such that  $c_i = m_i \circ e_i$ . Then  $c_{i+1}$  is the upper path in the diagram below.

$$\begin{array}{ccccccc} X & \xrightarrow{c} & BX & \xrightarrow{Be_i} & BE_i & \xrightarrow{Bm_i} & BW_i = W_{i+1} \\ & \searrow e_{i+1} & & & \nearrow & & \\ & & E_{i+1} & & & & \end{array} \quad (\text{A.1})$$

Notice that  $Bm_i$  is in  $\mathcal{M}$ , since  $B$  preserves  $\mathcal{M}$ -morphisms by assumption. Let  $e_{i+1}: X \rightarrow E_{i+1}$  be the  $\mathcal{E}$ -morphism of an  $(\mathcal{E}, \mathcal{M})$ -factorization of  $Be_i \circ c$ . We obtain an  $(\mathcal{E}, \mathcal{M})$ -factorization of  $c_{i+1}$ , since  $\mathcal{M}$ -morphisms are closed under composition. Thus  $e_{i+1}$  is the  $(i+1)$ -minimization of  $(X, c)$ .  $\square$

## B Proofs of Section 5

**Theorem 5.1** *Let  $a: LX \rightarrow X$  be an algebra.*

- (i) *Suppose that  $\mathcal{M}$  consists of monomorphisms, and suppose that the  $i$ -reachable part  $m_i: R_i \rightarrow X$  of  $(X, a)$  is an algebra morphism from an  $L$ -algebra  $(R_i, \varrho)$  to  $(X, a)$ . Then  $(R_i, \varrho)$  is the reachable part of  $(X, a)$ .*
- (ii) *In addition to the above assumptions, suppose  $\mathbf{D}$  is wellpowered, and  $L$  preserves morphisms in  $\mathcal{E}$ . Then the reachable part of any  $L$ -algebra exists, with carrier  $R_i$  for some ordinal number  $i$ .*
- (iii) *Suppose  $L$  preserves morphisms in  $\mathcal{E}$ , and  $m_i: R_i \rightarrow X$  is the  $i$ -reachable part of  $(X, a)$ . Then the  $\mathcal{M}$ -morphism of an  $(\mathcal{E}, \mathcal{M})$ -factorization of  $a \circ Lm_i: LR_i \rightarrow X$  is the  $(i+1)$ -reachable part of  $(X, a)$ .*

**Proof.** This follows directly by duality and Theorem 4.4: the factorization system  $(\mathcal{E}, \mathcal{M})$  on  $\mathbf{D}$  yields the factorization system  $(\mathcal{M}, \mathcal{E})$  on  $\mathbf{D}^{\text{op}}$ ,  $L$ -algebras in  $\mathbf{D}$  are  $L^{\text{op}}$ -algebras in  $\mathbf{D}^{\text{op}}$  and  $(i)$ -reachable parts in  $\mathbf{D}$  are  $(i)$ -minimizations in  $\mathbf{D}^{\text{op}}$ .

For item (iii), it may be helpful to see a direct proof. Let  $e_i$  be the  $\mathcal{E}$ -morphism such that  $m_i \circ e_i = a_i$ . Consider the following diagram, where the horizontal

path is  $a_{i+1}: V_{i+1} \rightarrow X$ , and  $m_{i+1}: R_{i+1} \rightarrow X$  is the  $\mathcal{M}$ -morphism of an  $(\mathcal{E}, \mathcal{M})$ -factorization of  $a \circ Lm_i$ :

$$V_{i+1} = LV_i \xrightarrow{Le_i} LR_i \xrightarrow{Lm_i} LX \xrightarrow{a} X \quad (B.1)$$

$\searrow \quad \nearrow m_{i+1}$   
 $R_{i+1}$

The morphism  $Le_i$  is in  $\mathcal{E}$ , since  $L$  preserves  $\mathcal{E}$ -morphisms by assumption. Since  $\mathcal{E}$ -morphisms compose, this yields a factorization of  $a_{i+1}$ , so that  $m_{i+1}$  is the  $(i+1)$ -reachable part of  $(X, a)$ .  $\square$

## C Proofs of Section 6

**Theorem 6.1** *Assume (A1) ... (A6) from the beginning of this section, and let  $(X, c)$  be a  $B$ -coalgebra. Let  $m: (R, \varrho) \rightarrow \overline{F}(X, c)$  be the reachable part of  $\overline{F}(X, c)$ . Take an (epi,mono)-factorization (in  $\text{coalg}(B)$ ) of the adjoint transpose  $m^\flat$  of  $m$ :*

$$(X, c) \xrightarrow{\quad} (E, \epsilon) \xrightarrow{\quad} \overline{G}(R, \varrho) \quad (3)$$

$\xrightarrow{m^\flat}$

Then  $(E, \epsilon)$  is the minimization of  $(X, c)$ .

**Proof.** Since  $L$  has an initial algebra  $(A, \alpha)$ ,  $m$  is the monic part of an (epi,mono)-factorization  $m \circ e: (A, \alpha) \rightarrow \overline{F}(X, c)$ . Because  $\overline{G}$  is a right adjoint, it maps colimits to limits, hence  $\overline{G}(A, \alpha)$  is a final coalgebra. Further, because  $G$  is a right adjoint and  $e$  is an epi,  $Ge$  is a mono (into the final coalgebra). Take an (epi,mono)-factorization of  $m^\flat$ , and compose with  $Ge$ :

$$(X, c) \xrightarrow{\quad} (E, \epsilon) \xrightarrow{\quad} \overline{G}(R, \varrho) \xrightarrow{Ge} \overline{G}(A, \alpha)$$

$\xrightarrow{m^\flat}$

Since monos are closed under composition, we have an (epi,mono)-factorization of the (unique) coalgebra morphism from  $(X, c)$  to the final  $B$ -coalgebra, i.e.,  $(E, \epsilon)$  is the minimization of  $(X, c)$ .  $\square$

## D Proofs of Section 7

**Lemma 7.1** *Let  $W: \text{Ord}^{\text{op}} \rightarrow \mathbf{C}$  be the final sequence of  $B$ , and  $V: \text{Ord} \rightarrow \mathbf{D}$  the initial sequence of  $L$ . There is a natural isomorphism  $\kappa: W \Rightarrow GV^{\text{op}}: \text{Ord}^{\text{op}} \rightarrow \mathbf{C}$  satisfying  $\kappa_{i+1} = \rho_{V_i} \circ B\kappa_i$  for all ordinals  $i$ .*

**Proof.** We define  $\kappa_i$  by transfinite induction. The successor step is given by the statement of the lemma. For a limit ordinal  $j$ , suppose we have an isomorphism  $\kappa_i: W_i \Rightarrow GV_i$  for all  $i < j$ . Since  $G$  is a right (contravariant) adjoint it maps colimits to limits, hence  $GV_j = G(\text{colim}_{i < j} V_i) = \lim_{i < j} GV_i$ . The aim is thus to find an isomorphism  $\kappa_j: \lim_{i < j} W_i \rightarrow \lim_{i < j} GV_i$ . By the inductive hypothesis we obtain cones

$$(\kappa_i \circ w_{j,i}: \lim_{i < j} W_i \rightarrow GV_i)_{i < j} \quad \text{and} \quad (\kappa_i^{-1} \circ Gv_{i,j}: \lim_{i < j} GV_i \rightarrow W_i)_{i < j}.$$

By the universal property of  $\lim_{i<j} GV_i$  and  $\lim_{i<j} W_i$ , we then obtain morphisms  $\kappa_j: \lim_{i<j} W_i \rightarrow \lim_{i<j} GV_i$  and  $\kappa_j^{-1}: \lim_{i<j} GV_i \rightarrow \lim_{i<j} W_i$ .

$$\begin{array}{ccc} \lim_{i<j} W_i & \xrightleftharpoons[\kappa_j^{-1}]{\kappa_j} & \lim_{i<j} GV_i \\ w_{j,i} \downarrow & & \downarrow Gv_{i,j} \\ W_i & \xrightleftharpoons[\kappa_i^{-1}]{\kappa_i} & GV_i \end{array}$$

The naturality squares as above are satisfied for each ordinal  $i$  with  $i \leq j$ , and it is not difficult to prove that  $\kappa_j$  and  $\kappa_j^{-1}$  are indeed each others inverse.  $\square$

For the proof of Theorem 7.2 (and Theorem 8.3), we will use the following result, which assumes functors  $B, L$ , a contravariant adjunction between  $F$  and  $G$  (as in Section 6) and a natural transformation  $\rho: BG \Rightarrow GL$ . Here  $\rho$  is not assumed to be an isomorphism; in this setting, the lifting  $\bar{F}: \text{coalg}(B)^{\text{op}} \rightarrow \text{alg}(L)$  of  $F$  is defined (as in Section 6), but, in general it does not have a right adjoint. As before, we denote the mate of  $\rho$  by  $\rho^\flat: LF \Rightarrow FB$ .

**Lemma D.1** *Let  $c: X \rightarrow BX$  be a coalgebra, and  $((\bar{F}c)_i: V_i \rightarrow FX)_{i \in \text{Ord}}$  the cocone over the initial sequence of  $L$  induced by  $\bar{F}(X, c)$ . There is a unique cone  $(c_i^\rho: X \rightarrow GV_i)_{i \in \text{Ord}}$  over  $GV^{\text{op}}$  such that*

$$c_{i+1}^\rho = (X \xrightarrow{c} BX \xrightarrow{Bc_i^\rho} BGV_i \xrightarrow{\rho_{V_i}} GLV_i).$$

For every  $i \in \text{Ord}$ , we have  $c_i^\rho = (\bar{F}c)_i^\flat$ .

**Proof.** Let  $(X, c)$  be a coalgebra. Uniqueness of the cone follows from the fact that when  $j$  is a limit ordinal, then  $GV_j = G\text{colim}_{i<j} V_i = \lim_{i<j} GV_i$ , where the latter equality holds since  $G$  is a right adjoint.

We show that

- (i)  $((\bar{F}c)_i^\flat: X \rightarrow GV_i)_{i \in \text{Ord}}$  is a cone over  $GV^{\text{op}}$ ;
- (ii) for all  $i$ , we have  $(\bar{F}c)_{i+1}^\flat = \rho_{V_i} \circ B(\bar{F}c)_i^\flat \circ c$ .

Since  $(c_i^\rho: X \rightarrow W_i)_{i \in \text{Ord}}$  is the unique cone with the property  $c_{i+1}^\rho = \rho_{V_i} \circ Bc_i^\rho \circ c$ , it then follows that  $c_i^\rho = (\bar{F}c)_i^\flat$  for all  $i$ .

- (i) Let  $i, j$  be ordinals with  $i \leq j$ . Since  $((\bar{F}c)_i: V_i \rightarrow FX)_{i \in \text{Ord}}$  is a cone over the initial sequence  $V$ , the triangle on the left-hand side commutes:

$$\begin{array}{ccc} V_j & & X \\ v_{i,j} \uparrow & \searrow (\bar{F}c)_j & \xrightarrow{(\bar{F}c)_i^\flat} GV_i \\ V_i & \xrightarrow{(\bar{F}c)_i} FX & \searrow (\bar{F}c)_j^\flat \\ & & GV_j \end{array} \quad \begin{array}{ccc} & & \\ & & \uparrow Gv_{i,j} \\ & & GV_j \end{array}$$

As a consequence, the triangle on the right-hand side commutes.

- (ii) By definition of  $\bar{F}$  we have  $\bar{F}(X, c) = Fc \circ \rho_X^\flat$ , and by definition of the cone  $((\bar{F}c)_i: V_i \rightarrow FX)_{i \in \text{Ord}}$  induced by  $\bar{F}(X, c)$ , the following commutes for any  $i$ :

$$\begin{array}{ccccccc}
 & & \xrightarrow{(\overline{F}c)_{i+1}} & & & & \\
 LV_i & \xrightarrow{L(\overline{F}c)_i} & LFX & \xrightarrow{\rho_X^\flat} & FBX & \xrightarrow{Fc} & FX
 \end{array} \quad (D.1)$$

Consider the following diagram.

$$\begin{array}{ccccccccc}
 & & & & \xrightarrow{(\overline{F}c)_{i+1}^\flat} & & & & \\
 X & \xrightarrow{\eta_X} & GFX & \xrightarrow{GFc} & GFBX & \xrightarrow{G\rho_X^\flat} & GLFX & \xrightarrow{GL(\overline{F}c)_i} & GLV_i \\
 & \searrow c & & \uparrow \eta_{BX} & & \uparrow \rho_{FX} & & \uparrow \rho_{V_i} & \\
 & & BX & \xrightarrow{B\eta_X} & BGF X & \xrightarrow{BG(\overline{F}c)_i} & BGV_i & & \\
 & & & & \xrightarrow{B(\overline{F}c)_i^\flat} & & & & 
 \end{array} \quad (D.2)$$

The upper crescent commutes by (D.1) and the definition of the adjoint transpose, and the lower crescent commutes as well by definition of the transpose. The left triangle and right square commute by naturality. The middle square is a standard property relating  $\rho$  and its mate, see, e.g., (the full version of) [15]. Commutativity of the outside of the diagram is the desired property.  $\square$

**Theorem 7.2** *Let  $(X, c)$  be a  $B$ -coalgebra. For any ordinal  $i$ , the following diagram commutes:*

$$\begin{array}{ccc}
 X & \xrightarrow{c_i} & W_i \\
 & \searrow (\overline{F}c)_i^\flat & \downarrow \kappa_i \\
 & & GV_i
 \end{array}$$

**Proof.** Let  $\kappa: W \Rightarrow GV^{\text{op}}$  be the isomorphism from Lemma 7.1. Consider the cone  $(c_i: X \rightarrow W_i)_{i \in \text{Ord}}$  induced by  $(X, c)$ . By naturality of  $\kappa$ , this yields a cone  $(\kappa_i \circ c_i: X \rightarrow GV_i)_{i \in \text{Ord}}$  over  $GV^{\text{op}}$ . Given an ordinal  $i$ , consider the following diagram:

$$\begin{array}{ccccc}
 X & \xrightarrow{c_{i+1}} & BW_i & \xrightarrow{\kappa_{i+1}} & GLV_i \\
 \downarrow c & & \parallel & & \uparrow \rho_{V_i} \\
 BX & \xrightarrow{Bc_i} & BW_i & \xrightarrow{B\kappa_i} & BGV_i
 \end{array}$$

The left square commutes by definition of  $(c_i)_{i \in \text{Ord}}$  and the right square commutes by Lemma 7.1. We have shown that  $(\kappa_i \circ c_i: X \rightarrow GV_i)_{i \in \text{Ord}}$  is a cone over  $GV^{\text{op}}$ , satisfying  $\kappa_{i+1} \circ c_{i+1} = \rho_{V_i} \circ B(\kappa_i \circ c_i) \circ c$ . By Lemma D.1, we obtain  $\kappa_i \circ c_i = (\overline{F}c)_i$  for all  $i$ .  $\square$

**Corollary 7.3** *Let  $(X, c)$  be a  $B$ -coalgebra. Let  $m_i^\flat: X \rightarrow GR_i$  be the transpose of the  $i$ -reachable part of  $\overline{F}(X, c)$ . Then the epic morphism  $e_i: X \rightarrow E_i$  of an (epi, mono)-factorization of  $m_i^\flat$  is the  $i$ -minimization of  $(X, c)$ . Further, if  $m_i: R_i \rightarrow FX$  is the reachable part of  $\overline{F}(X, c)$ , then  $e_i$  is the minimization of  $(X, c)$ .*



**Proof.** The arrow  $m_i: R_i \rightarrow FX$  is the  $i$ -reachable part of  $\bar{F}(X, c)$ , thus it is part of a factorization  $m_i \circ e' = (\bar{F}c)_i$  for some epi  $e': V_i \rightarrow R_i$ . Consider the diagram:

$$\begin{array}{ccccc}
 X & \xrightarrow{c_i} & W_i & & \\
 \downarrow & \searrow^{(\bar{F}c)_i} & \uparrow^{\kappa_i^{-1}} & & \\
 E_i & \xrightarrow{m_i^b} & GR_i & \xrightarrow{Ge'} & GV_i
 \end{array}$$

where the lower left triangle is an (epi,mono)-factorization of  $m_i^b$ . The middle triangle commutes by construction of  $m_i$ , and the upper right by Theorem 7.2. The arrow  $Ge'$  is mono, since  $e'$  is epi and  $G$  is a right adjoint. Hence we obtained an (epi,mono)-factorization of  $c_i$ , so the epic part is the  $i$ -minimization of  $(X, c)$ .

For the second part of the statement, suppose  $m_i$  is the reachable part of  $\bar{F}(X, c)$ , meaning in particular that there is an algebra  $(R_i, \varrho)$  on  $R_i$  turning  $m_i: (R_i, \varrho) \rightarrow \bar{F}(X, c)$  into an algebra morphism. Then the adjoint transpose  $m_i^b$  (in the lifted adjunction between  $\bar{F}$  and  $\bar{G}$ ) is a coalgebra morphism  $m_i^b: (X, c) \rightarrow \bar{G}(R_i, \varrho)$ . Consider the epic part of a factorization  $e_i: (X, c) \rightarrow (E_i, \epsilon)$  of this coalgebra morphism  $m_i^b$ . The underlying map  $e_i: X \rightarrow E_i$  is the epic part of the factorization of  $m_i^b$  (in  $\mathbf{C}$ ), hence, by the first part of the corollary, it is the  $i$ -minimization. Since  $e_i$  is a coalgebra morphism, by Theorem 4.4 it is the minimization of  $(X, c)$ .  $\square$

**Lemma 7.5** Suppose  $\mathbf{C}$  has pullbacks. Let  $e_i: X \rightarrow E_i$  be the  $i$ -minimization of a coalgebra  $c: X \rightarrow BX$ , and let  $r_{i,i+1}: R_i \rightarrow R_{i+1}$  be the arrow (see Section 5) from the  $i$ -reachable part  $m_i: R_i \rightarrow FX$  to the  $(i+1)$ -reachable part  $m_{i+1}: R_{i+1} \rightarrow FX$  of  $\bar{F}(X, c)$ . By Corollary 7.3,  $m_i^b = m' \circ e_i$  for some mono  $m'$ . Let  $P$  be the pullback of  $m'$  and  $Gr_{i,i+1}$ :

$$\begin{array}{ccc}
 X & \xrightarrow{m_{i+1}^b} & GR_{i+1} \\
 \searrow^{h} & \downarrow \lrcorner & \downarrow Gr_{i,i+1} \\
 P & \xrightarrow{\quad} & GR_{i+1} \\
 \downarrow e_i & & \downarrow \\
 E_i & \xrightarrow{m'} & GR_i
 \end{array}$$

There is a unique mediating morphism  $h$  as above. The epic part of an (epi,mono)-factorization of  $h$  is the  $(i+1)$ -minimization of  $(X, c)$ .

**Proof.** The arrow  $r_{i,i+1}$  satisfies  $m_{i+1} \circ r_{i,i+1} = m_i$ , so  $Gr_{i,i+1} \circ m_{i+1}^b = m_i^b$ , and since  $m_i^b = m' \circ e_i$ , we get  $m' \circ e_i = Gr_{i,i+1} \circ m_{i+1}^b$ . Hence, the unique morphism  $h: X \rightarrow P$  arises by the universal property of the pullback.

Pullbacks are stable under monomorphisms: since  $m'$  is a mono, the arrow  $P \rightarrow GR_{i+1}$  is a mono as well. Hence, an (epi,mono)-factorization of  $h$  yields, by composition, an (epi,mono)-factorization of  $m_{i+1}^b$ . The epic part of such a factorization is the  $(i+1)$ -minimization of  $(X, c)$ , by Corollary 7.3.  $\square$

Below Lemma 7.5, we gave a concrete presentation of the pullback, for the case of deterministic automata. Here we fill in missing details. By assumption,  $R_i, R_{i+1}$  are presented as subsets of  $2^X$ , i.e.,  $m_i: R_i \rightarrow 2^X$  and  $m_{i+1}: R_{i+1} \rightarrow 2^X$  are inclusion

maps. Since  $m_{i+1} \circ r_{i,i+1} = m_i$ , we have  $R_i \subseteq R_{i+1}$ , witnessed by the inclusion map  $r_{i,i+1}: R_i \rightarrow R_{i+1}$ . Hence  $Gr_{i,i+1} = 2^{r_{i,i+1}}: 2^{R_{i+1}} \rightarrow 2^{R_i}$  is given by  $2^{r_{i,i+1}}(C) = \{S \in R_i \mid r_{i,i+1}(S) \in C\} = C \cap R_i$ . Further, we have  $m_i^b(x) = \{S \in R_i \mid x \in S\}$ , and  $m': E_i \rightarrow 2^{R_i}$  is given by  $m'(Q) = \{S \in R_i \mid Q \subseteq S\}$ .

We start from a standard description of the pullback of  $m'$  and  $2^{r_{i,i+1}}$  in **Set** in the derivation below, as the set of pairs that are equated by  $m'$  and  $2^{r_{i,i+1}}$  (together with the projection maps to  $E_i$  and  $2^{R_{i+1}}$ ).

$$\begin{aligned} & \{(Q, C) \in E_i \times 2^{R_{i+1}} \mid m'(Q) = 2^{r_{i,i+1}}(C)\} \\ &= \{(Q, C) \in E_i \times 2^{R_{i+1}} \mid \{S \in R_i \mid Q \subseteq S\} = C \cap R_i\} \\ &= \{(Q, C) \in E_i \times 2^{R_{i+1}} \mid \forall S \in R_i : S \in C \text{ iff } Q \subseteq S\} \\ &\cong \{(Q, C) \in E_i \times 2^{R_{i+1}} \mid C \subseteq R_{i+1} \setminus R_i\}. \end{aligned}$$

The latter set is the characterization of the pullback  $P$  given in Section 7. The isomorphism, up-down is given by  $(Q, C) \mapsto (Q, \{S \in C \mid S \in R_{i+1} \setminus R_i\})$ , and down-up by  $(Q, C) \mapsto (Q, \{S \in R_i \mid Q \subseteq S\} \cup C) = (Q, m'(Q) \cup C)$ . It is easy to check that these maps indeed form each others inverse. By the isomorphism, the maps  $p_1: P \rightarrow E_i$  and  $p_2: P \rightarrow 2^{R_{i+1}}$  of the pullback are given by  $p_1(Q, C) = Q$  and  $p_2(Q, C) = m'(Q) \cup C$ . The map  $h: X \rightarrow P$  given in Section 7 trivially satisfies  $p_1 \circ h = e_i$ . Further,

$$\begin{aligned} p_2 \circ h(x) &= p_2(e_i(x), \{S \in R_{i+1} \setminus R_i \mid x \in S\}) \\ &= m'(e_i(x)) \cup \{S \in R_{i+1} \setminus R_i \mid x \in S\} \\ &= m_i^b(x) \cup \{S \in R_{i+1} \setminus R_i \mid x \in S\} \\ &= \{S \in R_i \mid x \in S\} \cup \{S \in R_{i+1} \setminus R_i \mid x \in S\} \\ &= m_{i+1}^b(x) \end{aligned}$$

which means that  $h$  is indeed the mediating map.

## E Proofs of Section 8

**Theorem 8.1** *Suppose that  $\mathbf{C}$  and  $\mathbf{D}$  have (epi,mono)-factorization systems. Let  $c: X \rightarrow BTX$  be a coalgebra, and let  $m: (R, \varrho) \rightarrow \overline{F}_\alpha(X, c)$  be the reachable part of  $\overline{F}_\alpha(X, c)$ . Then the epic part of an (epi,mono)-factorization (in  $\mathbf{C}$ ) of the transpose  $m^b: X \rightarrow GR$  is the language quotient of  $(X, c)$ .*

**Proof.** Let  $(A, \alpha)$  be the initial algebra (which exists by assumption) and let  $s: (A, \alpha) \rightarrow \overline{F}_\alpha(X, c)$  be the unique algebra homomorphism. The reachable part  $m: (R, \varrho) \rightarrow \overline{F}_\alpha(X, c)$  is the monic morphism of an (epi,mono) factorization  $m \circ e = s$  (in  $\mathbf{alg}(L)$ ). We get  $s^b = Ge \circ m^b$ , and since  $e$  is epi and  $G$  is a right (contravariant) adjoint,  $Ge$  is mono. Thus, an (epi,mono)-factorization of  $m^b$  yields an (epi,mono)-factorization of  $Ge \circ m^b = s^b$ , by composition. Its epic part is, by definition, the language quotient of  $(X, c)$ .  $\square$

**Theorem 8.3** *Let  $c: X \rightarrow BTX$  be a coalgebra, and  $((\overline{F}_\alpha c)_i: V_i \rightarrow FX)_{i \in \mathbf{Ord}}$  the cocone over the initial sequence of  $L$  induced by  $\overline{F}_\alpha(X, c)$ . For any  $i$ , we have*

$\bar{c}_i = (\bar{F}_\alpha c)_i^\flat$ . Further, let  $m_i: R_i \rightarrow FX$  be the  $i$ -reachable part of  $\bar{F}_\alpha(X, c)$ . Then the epic morphism of an (epi,mono)-factorization of the transpose  $m_i^\flat: X \rightarrow GR_i$  is the  $i$ -language quotient of  $(X, c)$ .

**Proof.** The natural transformations  $\rho: BG \Rightarrow GL$  and  $\alpha: TG \Rightarrow G$  compose to a natural transformation  $\rho \circ B\alpha: BTG \Rightarrow GL$ . Now, the cone  $(\bar{c}_i)_{i \in \text{Ord}}$  is the same as the cone  $(c_i^{\rho \circ B\alpha})_{i \in \text{Ord}}$  of Lemma D.1 (instantiating  $B$  and  $\rho$  in the lemma respectively to  $BT$  and  $\rho \circ B\alpha$ ; then the functor  $\bar{F}$  in the lemma coincides with  $\bar{F}_\alpha$  from Section 8). By the lemma, we obtain  $\bar{c}_i = (\bar{F}_\alpha c)_i^\flat$  for all  $i$ .

For the second part of the statement, let  $m_i: R_i \rightarrow FX$  be the  $i$ -reachable part of  $\bar{F}(X, c)$ , and let  $e_i$  be the epi such that  $m_i \circ e_i = (\bar{F}_\alpha c)_i$ . Then  $\bar{c}_i = (\bar{F}_\alpha c)_i^\flat = Ge_i \circ m_i^\flat$ , and since  $e_i$  is epi and  $G$  is a right (contravariant) adjoint,  $Ge_i$  is mono. Thus, an (epi,mono)-factorization of  $m_i^\flat$  yields an (epi,mono)-factorization of  $\bar{c}_i$ , by composition. Its epic part is, by definition, the  $i$ -language quotient of  $(X, c)$ .  $\square$

# A predicate/state transformer semantics for Bayesian learning

Bart Jacobs and Fabio Zanasi

*Radboud University Nijmegen, The Netherlands*

---

## Abstract

This paper establishes a link between Bayesian inference (learning) and predicate and state transformer operations from programming semantics and logic. Specifically, a very general definition of backward inference is given via first applying a predicate transformer and then conditioning. Analogously, forward inference involves first conditioning and then applying a state transformer. These definitions are illustrated in many examples in discrete and continuous probability theory and also in quantum theory.

*Keywords:* Inference, learning, Bayes, Kleisli category, effectus, predicate transformer, state transformer

---

## 1 Introduction

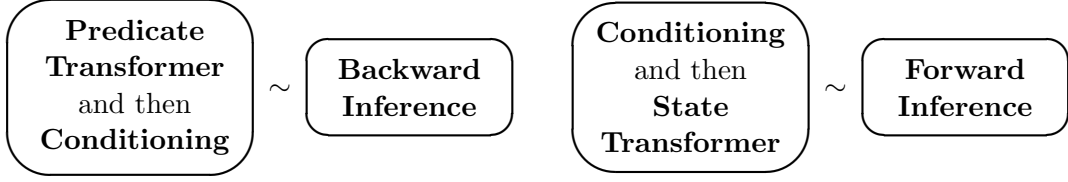
Increasingly probabilistic programs are used to describe problems in Bayesian inference ([2]), see *e.g.* [10,19,4,1,21]. The term ‘inference’ is used for what is informally best called: learning<sup>1</sup>. Learning involves updating one’s knowledge, in the light of certain evidence, typically given via the validity of a certain predicate (which may be a fuzzy one). In this situation one represents knowledge in terms of likelihoods, via a probability distribution (in the discrete case) or a probability measure (in the continuous case). Updating one’s knowledge then involves computing a conditional distribution/measure.

Now that the overlap between the (probabilistic) programming community and the Bayesian community is growing, a merging of concepts and techniques can be expected. This paper is an example. It shows how the notions of predicate and state transformer from programming languages semantics ([7]) can be used in precisely defining two fundamental notions of learning: backward and forward inference. A conditioning operation, which makes a certain distribution/measure depend on a predicate, also plays a role. In a nutshell, the correspondence can be summarised

---

<sup>1</sup> The Bayesian community associates learning to various tasks. A prominent learning task is finding out what the topology of a network is, based on (in)dependence relations, starting from a big joint distribution.

as follows.



This connection hopefully works as an *Aha Erlebnis*, giving a sudden insight. Indeed, predicate transformers work backwards, from predicates on post-states to predicates on pre-states. This is precisely what is at stake in backward inference — as we will demonstrate. Similarly, state transformers work in a forward direction, which is what happens in forward learning.

Strictly speaking, the main contribution of this paper is only one definition, namely of (backward and forward) inference, see Definition 2.1. Contrarily to traditional approaches, our formulation is not tied to the probabilistic setting, but works in the context of any *effectus*, that is a categorical notion embracing a wide spectrum of computational models, both classical, probabilistic and also quantum, see [11,5]. Within the theory of effectuses, predicate and state transformers are well-defined, and predicates (or effects) and states can be nicely organised in state-and-effect triangles, which connect predicates and states via a (dual) adjunction (1), see also [12]. Intriguingly, these triangles correspond to what physicists call the duality between states and effects, referring to the opposite directions in the work of Schrödinger and Heisenberg on quantum foundations. Within this effectus context one can also describe normalisation and conditioning of states in an abstract manner (see [13,5]). Therefore, we believe effectuses form the right setting for developing a general approach to inference.

Still, precisely recognising what is what in this setting is a subtle matter. For instance, what is a predicate, at the abstract level? Traditionally in probability theory ‘events’ are used as predicates. Formally they are subsets of the sample space, corresponding to ‘sharp’ predicates on this space. More generally, ‘fuzzy’ predicates are considered; they are functions taking values in the unit interval  $[0, 1]$ . The sharp predicates can then be characterised as the ones taking values in the Boolean subset  $\{0, 1\} \subseteq [0, 1]$ . In discrete probability every distribution is at the same time a fuzzy predicate. This blurs the picture — the confusion between states and predicates is particularly evident in Bayesian network representations, where nodes may play both roles. In continuous probability there is, in principle, a clear distinction between states (probability measures) and predicates (measurable functions to  $[0, 1]$ ). But again, things easily get mixed up, when a state/measure is given by a probability distribution function (pdf), which looks very much like a predicate. The framework of effectus theory helps in this respect, since it gives a clear distinction between states, as maps of the form  $1 \rightarrow X$ , and predicates, as maps  $X \rightarrow 1 + 1$ . Only when this perspective is recognised, the role of predicate and state transformers becomes clear. It is for this reason that we think it is justified to dedicate an entire paper to elaborating and explaining a single definition.

The paper is organised as follows. We first introduce the notions of backward and forward inference in terms of predicate and state transformers and show some

basic properties. Then, we concentrate on illustrating the impact and power of our definition in many situations. We show what our abstract setting translates to in discrete and continuous probability theory and also (briefly) in quantum theory. We elaborate many examples of computations of how inference works, and what it produces. Of special interest is the application of our definition of inference in Bayesian networks. It is shown that the forward/backward distinction can be used flexibly, and can describe what inference means at different points in the network.

## 2 Backward and forward inference, abstractly

In this section we describe our abstract set up for inference, both in a backward and forward manner. This works in the setting of an *effectus*: briefly, this is a category with finite coproducts  $(+, 0)$  and a final object  $1$ , such that certain diagrams are pullbacks and certain maps are jointly monic. By virtue of these basic requirements, an effectus is able to capture some basic aspects of quantum computation, with probabilistic computation as special case, see [11, 5].

*States* in an effectus  $\mathbf{C}$  are maps of the form  $1 \rightarrow X$  and *predicates* are maps  $X \rightarrow 2 = 1 + 1$ . The set of states  $\text{Stat}(X)$  of an object  $X$  form a convex set, and the set of predicates  $\text{Pred}(X)$  on  $X$  form an effect module. States and predicates give rise to a ‘state-and-effect triangle’ of the form:

$$\begin{array}{ccc}
 \mathbf{EMod}^{\text{op}} & \xrightleftharpoons{\top} & \mathbf{Conv} \\
 \text{Pred} = \text{Hom}(-, 2) \swarrow & & \searrow \text{Stat} = \text{Hom}(1, -) \\
 & \mathbf{C} &
 \end{array} \tag{1}$$

We refer to [11] for details about effect modules and convex sets. In the current setting we need the *predicate transformer*  $f^* = \text{Pred}(f)$  and *state transformer*  $f_* = \text{Stat}(f)$  operations associated with a map  $f: X \rightarrow Y$  in the base category  $\mathbf{C}$ . They are given by pre- and post-composition:

$$\text{Pred}(X) \xleftarrow{f^* = (-) \circ f} \text{Pred}(Y) \qquad \text{Stat}(X) \xrightarrow{f_* = f \circ (-)} \text{Stat}(Y)$$

In concrete examples of effectuses states are distributions — in the Kleisli category of the distribution monad — or probability measures — in the Kleisli category of the Giry monad — or just states — in  $C^*$ - or  $W^*$ -algebras. We will understand states as descriptions of our state of knowledge. Given a predicate  $p$  and a state  $\omega$  on the same object  $X$  two definitions are of interest:

$$\omega \models p := p \circ \omega \qquad \text{and} \qquad \omega|_p, \text{ the conditional state on } X. \tag{2}$$

The expression  $\omega \models p$  describes the validity, or expected value, of the predicate in the state  $\omega$ . Typically its value is in the unit interval  $[0, 1]$ . If this validity  $\omega \models p$  is non-zero, then the conditional state  $\omega|_p$  exists. It is the updated state of knowledge after observing ‘evidence’  $p$ . In each of the above concrete examples of states we can define such conditional states (see below). In fact, conditioning can be defined abstractly in the theory of effectuses, using ‘assert’ maps, see [5, Example 58], but we don’t need such a level of abstraction here.

We now distinguish two forms of inference (learning).

**Definition 2.1** *Backward inference*  $\omega|_{f^*(p)}$  involves first applying a predicate transformer and then computing a conditional. This applies in situations of the form:

$$1 \xrightarrow{\omega} X \xrightarrow{f} Y \xrightarrow{q} 1 + 1 \quad (3)$$

More explicitly, one first applies the predicate transformer  $f^*$  to the predicate  $q$  on  $Y$ , and then computes the backwardly inferred conditional state  $\omega|_{f^*(q)}$  on  $X$ .

*Forward inference*  $f_*(\omega|_p)$  is first computing a conditional and then applying a state transformer. This works in a situation:

$$\begin{array}{ccc} 1 & \xrightarrow{\omega} & X \xrightarrow{f} Y \\ & & \downarrow p \\ & & 1 + 1 \end{array} \quad (4)$$

In this case the conditional state on  $X$  is  $\omega|_p$ , and applying the state transformer  $f_*$  gives the forwardly inferred state  $f_*(\omega|_p)$  on  $Y$ .

In the trivial case where the map  $f$  is the identity there is no difference between backward and forward inference. Inference then just involves updating a state (of knowledge). Notice that in backward inference we use a predicate on the codomain of the map  $f$ , namely  $q$ , and update our knowledge about the state on  $f$ 's domain  $X$ . In forward inference we use a predicate on the domain of  $f$ , namely  $p$ , and use it to infer more about the state on  $f$ 's codomain  $Y$ . This may also be called ‘evidence propagation’.

In the situation (4) we have the following Galois style equalities for validity:

$$(\omega \models f^*(q)) = q \circ f \circ \omega = (f_*(\omega) \models q).$$

In general, there are very few ‘nice’ algebraic properties for conditional states. For instance, we do have  $f_*(\omega|_{f^*(q)}) = (f \circ \omega)|_q$ , but only for the special case where the map  $f$  is ‘pure’. The latter means for instance in a Kleisli category that the map comes from the underlying category.

In the remainder of this paper we shall illustrate these forms of inference via several examples, involving various kinds of computation, and including Bayesian networks where the above map  $f$  in (3) and (4) arises from a graph (network of conditional probability tables). The composition notation ‘ $\circ$ ’ used above looks deceptively simple, but will each time be interpreted differently in different categories. This leads to various concrete forms of inference which are all instances of the same pattern.

### 3 Inference with discrete probability

We shall write  $\mathcal{D}$  for the discrete probability monad on the category **Set** of sets and functions. The set  $\mathcal{D}(X)$  contains the finite discrete probability distributions

$\omega$  over  $X$  which we write as formal convex combinations:

$$\omega = r_1 |x_1\rangle + \cdots + r_n |x_n\rangle \quad \text{where} \quad \begin{cases} x_1, \dots, x_n \in X \\ r_1, \dots, r_n \in [0, 1] \text{ with } \sum_i r_i = 1. \end{cases}$$

The ‘ket’ notation  $|x\rangle$  is meaningless syntactic sugar, used to distinguish elements  $x \in X$  from their occurrence in such formal convex sums. Notice that such  $\omega \in \mathcal{D}(X)$  can be identified with functions  $\omega: X \rightarrow [0, 1]$  with finite support  $\text{supp}(\omega) = \{x \in X \mid \omega(x) \neq 0\}$  and with  $\sum_{x \in X} \omega(x) = 1$ . This function-description is often more convenient.

We shall write  $\mathcal{Kl}(\mathcal{D})$  for the Kleisli category of the distribution monad  $\mathcal{D}$ . Its objects are sets  $X$ , and its morphisms  $X \rightarrow Y$  are stochastic matrices, in the form of functions  $X \rightarrow \mathcal{D}(Y)$ .

We will see later (Section 3.1) how Bayesian networks can be seen as certain arrows of  $\mathcal{Kl}(\mathcal{D})$ . For this interpretation, it is of importance that  $\mathcal{Kl}(\mathcal{D})$  forms a symmetric monoidal category, with the following ingredients. The monoidal product  $\otimes$  is defined on objects as the cartesian product  $\times$  in **Set**, with unit 1. On arrows  $f: A \rightarrow X$  and  $g: B \rightarrow Y$ , it is defined as

$$f \otimes g := \left( A \times B \xrightarrow{f \times g} \mathcal{D}(X) \times \mathcal{D}(Y) \xrightarrow{\text{dst}} \mathcal{D}(X \times Y) \right)$$

where the map  $\text{dst}$  sends a pair  $(\rho, \sigma) \in \mathcal{D}(X) \times \mathcal{D}(Y)$  to the distribution in  $\mathcal{D}(X \times Y)$  given by  $(x, y) \mapsto \rho(x) \cdot \sigma(y)$ . The symmetry  $\text{tw}_{X,Y}$  on  $X \times Y$  is the lifting to  $\mathcal{Kl}(\mathcal{D})$  of the isomorphism  $X \times Y \xrightarrow{\cong} Y \times X$  in **Set**; we will omit the subscript when  $X$  and  $Y$  are clear from the context.

We now turn to the description of states and predicates in  $\mathcal{Kl}(\mathcal{D})$ . Notice that states  $\omega: 1 \rightarrow X$  in  $\mathcal{Kl}(\mathcal{D})$  can be identified with distributions  $\omega \in \mathcal{D}(X)$ . Since  $\mathcal{D}(2) \cong [0, 1]$  we can identify predicates  $X \rightarrow 2 = 1 + 1$  in  $\mathcal{Kl}(\mathcal{D})$  with functions  $X \rightarrow [0, 1]$ , that is, with fuzzy predicates. We will often make both identifications when emphasising the role of states and predicates in a computation.

Given a Kleisli map  $f: X \rightarrow \mathcal{D}(Y)$ , a state  $\omega \in \mathcal{D}(X)$  and a predicate  $q \in [0, 1]^Y$  we have the following descriptions for state and predicate transformation. They arise from unravelling (Kleisli) composition in  $\mathcal{Kl}(\mathcal{D})$ .

$$\begin{aligned} f_*(\omega) &:= \sum_{y \in Y} \left( \sum_{x \in X} f(x)(y) \cdot \omega(x) \right) |y\rangle \\ f^*(q)(x) &:= \sum_{y \in Y} f(x)(y) \cdot q(y). \end{aligned} \tag{5}$$

For a distribution  $\omega \in \mathcal{D}(X)$  and a predicate  $p \in [0, 1]^X$  on the same set  $X$  we define the validity  $\omega \models p$  in  $[0, 1]$  as:

$$\omega \models p := \sum_{x \in X} \omega(x) \cdot p(x). \tag{6}$$

If this validity  $\omega \models p$  is non-zero, then the conditional state  $\omega|_p \in \mathcal{D}(X)$  is given as



formal convex sum:

$$\omega|_p := \sum_{x \in X} \frac{\omega(x) \cdot p(x)}{\omega \models p} |x\rangle. \quad (7)$$

We shall describe a familiar medical test example in the current setting. We use the following notational convention. We write a letter  $D$  for a certain disease, which is represented as a two-element set  $2_D = \{d, d^\perp\}$ , where the element  $d$  represents occurrence of the disease, and  $d^\perp$  represents non-occurrence. A distribution over  $2_D$  is, *e.g.*, of the form  $\frac{1}{4}|d\rangle + \frac{3}{4}|d^\perp\rangle$ , when describing that the disease occurs with probability  $\frac{1}{4}$ . Similar we write  $2_T$  for a (positive) test, where  $2_T = \{t, t^\perp\}$ . For each such set  $2_A = \{a, a^\perp\}$  we write  $A?: 2_A \rightarrow [0, 1]$  for the sharp predicate given by  $A?(a) = 1$  and  $A?(a^\perp) = 0$ .

Consider the following situation in the Kleisli category  $\mathcal{KL}(\mathcal{D})$ .

$$1 \xrightarrow{\omega} 2_D \xrightarrow{s} 2_T \quad \text{with} \quad \begin{cases} \omega = \frac{1}{100}|d\rangle + \frac{99}{100}|d^\perp\rangle \\ s(d) = \frac{9}{10}|t\rangle + \frac{1}{10}|t^\perp\rangle \\ s(d^\perp) = \frac{1}{20}|t\rangle + \frac{19}{20}|t^\perp\rangle. \end{cases}$$

The state  $\omega$  describes the *a priori* probability of 1% that someone has the disease. The map  $s$  describes the sensitivity of the test: when someone has the disease, the test will be positive in 90% of the cases, and when someone does not have the disease there is still a 5% chance that the test is positive.

A basic question is: what is the chance that I have the disease if I test positive? We formalise this by adding the predicate  $T?: 2_T \rightarrow [0, 1]$ , which expresses that there is a positive test. We then compute consecutively the predicate  $s^*(T?): 2_D \rightarrow [0, 1]$ , the validity  $\omega \models s^*(T?)$  and the inferred conditional state  $\omega|_{s^*(T?)}$ . We use formulas (5), (6), and (7) for backward inference from Definition 2.1:

$$\begin{aligned} s^*(T?)(d) &= \frac{9}{10} \cdot 1 + \frac{1}{10} \cdot 0 \\ &= \frac{9}{10} \\ s^*(T?)(d^\perp) &= \frac{1}{20} \cdot 1 + \frac{19}{20} \cdot 0 \\ &= \frac{1}{20} \\ \omega \models s^*(T?) &= \frac{1}{100} \cdot \frac{9}{10} + \frac{99}{100} \cdot \frac{1}{20} \\ &= \frac{9}{1000} + \frac{99}{2000} \\ &= \frac{117}{2000} \\ \omega|_{s^*(T?)} &= \frac{2000}{117} \cdot \left( \frac{1}{100} \cdot \frac{9}{10} |d\rangle + \frac{99}{100} \cdot \frac{1}{20} |d^\perp\rangle \right) \\ &= \frac{18}{117} |d\rangle + \frac{99}{117} |d^\perp\rangle. \end{aligned} \quad (8)$$

Hence after a positive test the chance that I have the disease is  $\frac{18}{117} \sim 15\%$ . This is an instance of backward inference, where an observation on the codomain (the test outcome) changes the state of knowledge about the domain (the disease occurrence). Of course, standard Bayesian methods will arrive at the same outcome. The point is that we can describe these methods here in a uniform, abstract manner via calculations in (Kleisli) categories.

We briefly describe a forward example. Suppose that I know that the chance of having this disease is half as likely for me, for instance because I belong to a particular age group. We model this via the predicate  $p: 2_D \rightarrow [0, 1]$  given by  $p(d) = \frac{1}{2}$  and  $p(d^\perp) = 1$ . We would like to learn what the probability is of getting a positive test under these circumstances.

We take a step back, and ask ourselves: what is the probability of getting a positive test in general — without the adapted likelihood. This probability is computed via the state transformer  $s_*$  from (5) — that is, via Kleisli composition in  $\mathcal{KL}(\mathcal{D})$  as:

$$\begin{aligned} s_*(\omega) &= \left(\frac{1}{100} \cdot \frac{9}{10} + \frac{99}{100} \cdot \frac{1}{20}\right) |t\rangle + \left(\frac{1}{100} \cdot \frac{1}{10} + \frac{99}{100} \cdot \frac{19}{20}\right) |t^\perp\rangle \\ &= \frac{117}{2000} |t\rangle + \frac{1883}{2000} |t^\perp\rangle. \end{aligned}$$

For forward inference we first compute the conditional state  $\omega|_p$  and then push it forward to a state  $s_*(\omega|_p)$  on  $2_T$ .

$$\begin{aligned} \omega \models p &= \frac{1}{100} \cdot \frac{1}{2} + \frac{99}{100} \cdot 1 \\ &= \frac{199}{200} \\ \omega|_p &= \frac{200}{199} \cdot \left(\frac{1}{100} \cdot \frac{1}{2} |d\rangle + \frac{99}{100} \cdot 1 |d^\perp\rangle\right) \\ &= \frac{1}{199} |d\rangle + \frac{198}{199} |d^\perp\rangle \\ s_*(\omega|_p) &= \left(\frac{1}{199} \cdot \frac{9}{10} + \frac{198}{199} \cdot \frac{1}{20}\right) |t\rangle + \left(\frac{1}{199} \cdot \frac{1}{10} + \frac{198}{199} \cdot \frac{19}{20}\right) |t^\perp\rangle \\ &= \frac{216}{3980} |t\rangle + \frac{3764}{3980} |t^\perp\rangle. \end{aligned}$$

Hence, upon knowing that I have a reduced (halved) risk, my chance of getting a positive test goes down from  $\frac{117}{2000} \sim 5.8\%$  to  $\frac{216}{3980} \sim 5.4\%$ . The impact is limited, because I only have a very small chance of having the disease in the first place — and the false positive probability of the test is 5%.

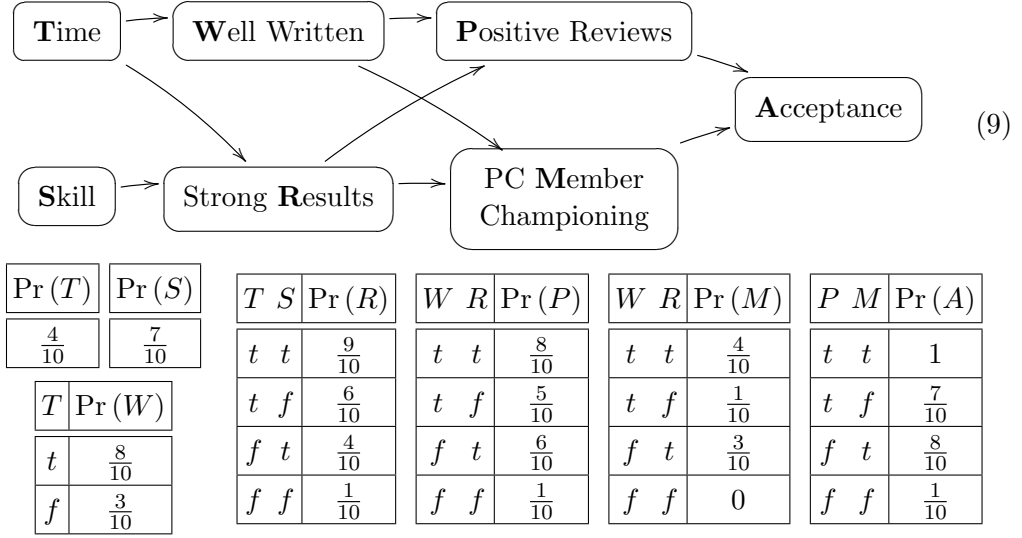
By imposing the predicate  $p$  on the disease state  $\omega$  we adapt the influence of the state  $\omega$  on the outcome. This may be useful for *counterfactual* reasoning, see [17]. In this way one can test to what extend a conclusion depends on certain initial states. For instance, if a particular conclusion is reached starting in a state where 70% of the participants is female, then by imposing an additional predicate on this state that changes the gender percentage, one can check if the same conclusion is reached.

### 3.1 Inference in a Bayesian network

Bayesian networks are graph-like structures, widely-adopted for the representation of probabilistic relationships between random events. They are usually depicted as directed acyclic graphs with nodes standing for random variables and edges indicating causal dependencies between them. Inference tasks are one of the fundamental uses of these networks. They are typically performed by updating a single node-event and then propagating the information to the rest of the network. Computing the inference typically goes through a repeated use of the Bayes' rule for conditional probability, see *e.g.* [16,18,17,2].

In this subsection we show how our abstract account of inference instantiates to the case of Bayesian networks. Our approach predicts the same outcomes as traditional Bayesian inference, but also improves it in two ways. First, it is more flexible and compositional, as it allows to focus on single nodes in the same way as on bigger portions of the network, with the same methodology. Second, it is more structured, in the sense that the computations that would require the use of Bayes' rule are carried out by the categorical machinery — essentially, by composition of arrows in a category.

In order to illustrate this picture, we will use as a running example the situation of a scientist that wants to publish a paper at a conference. The specification for the corresponding Bayesian network consists of a graph together with conditional probability tables.



The initial conditions of the example estimate whether there is enough time available to prepare the paper (the variable  $T$ ) and whether the scientist is sufficiently skilled to do the necessary research ( $S$ ). The results that the scientist is able to obtain ( $R$ ) depend both on the time and the skill, while how well the paper reads only depends on the time. Both results and readability have an influence on whether the reviews will be positive ( $P$ ), but results will be more relevant. Similarly, these two factors may lead a PC member to enthusiastically endorse the paper ( $M$ ), independently of what the reviewers say, although this possibility is quite rare. Finally, acceptance ( $A$ ) is influenced by the reviews and by the possible endorsement of a PC member.

In order to study inference in this example, we first need to formulate it in more categorical terms. We shall express our Bayesian network (9) as an arrow in the Kleisli category  $\mathcal{Kl}(\mathcal{D})$  of the distribution monad  $\mathcal{D}$ . First, each node  $N$  of the graph, say with  $k$  incoming edges from nodes  $N_1, N_2, \dots, N_k$ , is associated with an arrow  $N: 2^k \rightarrow k$  in  $\mathcal{Kl}(\mathcal{D})$ , which we conveniently write using the same labeling convention for the elements of 2 as in the disease example:

$$2_{N_1} \otimes 2_{N_2} \otimes \dots \otimes 2_{N_k} \xrightarrow{N} 2_N.$$

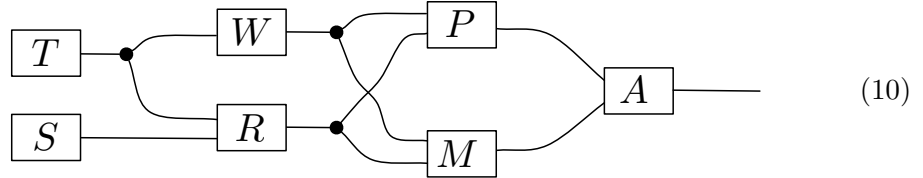
The probability distributions defining  $N$  are given according to the probability table of the node. For instance, the Kleisli map  $A: 2_P \otimes 2_M \rightarrow 2_A$  for acceptance is defined by:

$$\begin{aligned} (p, m) &\mapsto 1 |a\rangle & (p, m^\perp) &\mapsto \frac{7}{10} |a\rangle + \frac{3}{10} |a^\perp\rangle \\ (p^\perp, m) &\mapsto \frac{8}{10} |a\rangle + \frac{2}{10} |a^\perp\rangle & (p^\perp, m^\perp) &\mapsto \frac{1}{10} |a\rangle + \frac{9}{10} |a^\perp\rangle. \end{aligned}$$

Another example is the initial map  $T: 1 \rightarrow 2_T$  for the time node, which amounts to the distribution  $\frac{4}{10} |t\rangle + \frac{6}{10} |t^\perp\rangle$  in  $\mathcal{D}(2_T) \cong [0, 1]$ . In order to recover the whole network (9), one pastes node-arrows together using the symmetric monoidal structure of  $\mathcal{KL}(\mathcal{D})$ , which we recalled in the beginning of this section. Nodes in (9) that have multiple outgoing edges are modeled by composing the corresponding arrow  $2^k \rightarrow 2$  with the pairing map  $\delta: 2 \rightarrow 2^2$  defined by  $x \mapsto 1 |(x, x)\rangle$ . The Bayesian network (9) in its entirety is then expressed as the following arrow in  $\mathcal{KL}(\mathcal{D})$ , where for simplicity we omit the subscripts naming the elements of each copy of 2.

$$\begin{array}{c} \begin{array}{ccc} 1 & \xrightarrow{T \otimes S} & 2 \otimes 2 \\ & \delta \otimes \text{id} \downarrow & \\ & 2 \otimes 2 \otimes 2 & \xrightarrow{W \otimes R} & 2 \otimes 2 \end{array} \\ \begin{array}{ccc} & & \uparrow \delta \otimes \delta \\ & (2 \otimes 2) \otimes (2 \otimes 2) & \\ & \uparrow \text{id} \otimes \text{tw} \otimes \text{id} & \\ & (2 \otimes 2) \otimes (2 \otimes 2) & \xrightarrow{P \otimes M} & 2 \otimes 2 \xrightarrow{A} & 2 \end{array} \end{array}$$

We have written the “structural” arrows vertically. A more insightful representation of the same arrow can be given using the graphical language of string diagrams [20], with  $2^k$  depicted as a bundle of  $k$  wires and  $\delta$  as  $\bullet \curvearrowright$ . The result almost resembles the original network.



It may be calculated<sup>2</sup> that the entire arrow  $1 \rightarrow 2$  in (10) amounts to the distribution  $0.48 |a\rangle + 0.52 |a^\perp\rangle$  in  $\mathcal{D}(2) \cong [0, 1]$ . In words: given 40% of chances that the scientist has enough time at disposal and 70% of chances of being adequately skilled, the odds of having a paper accepted at the conference is  $\sim 48\%$ .

We now have everything in place to instantiate our framework for inference. As this example is more elaborated than the previous ones, it gives us the possibility to explore the situation in which knowledge update only involves a segment of the computation, namely  $f$  or  $g$  in the following partitioned version of (10).

$$1 \xrightarrow{\omega = \begin{array}{|c|} \hline T \\ \hline S \\ \hline \end{array}} 2 \otimes 2 \xrightarrow{f = \begin{array}{|c|} \hline W \\ \hline R \\ \hline \end{array}} 2 \otimes 2 \xrightarrow{g = \begin{array}{|c|} \hline P \\ \hline M \\ \hline \end{array}} 2 \xrightarrow{A} 2$$

<sup>2</sup> For simplicity, here and in the next calculations we approximate distribution values to two decimal digits.

In order to formulate a backward inference question, we follow the recipe (3) and introduce a predicate  $A?: 2_A \rightarrow [0, 1]$  that tests for acceptance of the paper. It is a sharp predicate, defined by  $A?(a) = 1$  and  $A?(a^\perp) = 0$ .

First we compute  $\omega|_{(g \circ f)^*(A?)}$ , that is, the odds that the accepted paper actually was submitted by a scientist with an adequate amount of time and skill to concoct it.

$$\begin{aligned} \omega &= 0.28 |t, s\rangle + 0.12 |t, s^\perp\rangle + 0.42 |t^\perp, s\rangle + 0.18 |t^\perp, s^\perp\rangle \\ (g \circ f)^*(A?) &= \begin{cases} (t, s) \mapsto 0.67 & (t, s^\perp) \mapsto 0.58 \\ (t^\perp, s) \mapsto 0.40 & (t^\perp, s^\perp) \mapsto 0.29 \end{cases} \\ \omega \models (g \circ f)^*(A?) &= 0.48 \\ \omega|_{(g \circ f)^*(A?)} &= \sum_{x \in 2_T \otimes 2_S} \frac{\omega(x) \cdot (g \circ f)^*(A?)(x)}{0.48} |x\rangle \\ &= 0.39 |t, s\rangle + 0.15 |t, s^\perp\rangle + 0.35 |t^\perp, s\rangle + 0.11 |t^\perp, s^\perp\rangle \end{aligned}$$

We observe that, after finding out that the paper has been accepted, the chances that the scientist had both sufficient time and skill rise from 28% to 39%.

As a second example, we shift the attention from the author to the paper itself. The following state on  $2_W \otimes 2_R$  expresses the chances that an accepted paper was actually well written and contained strong scientific results. Note that it mixes state and predicate transformers to bind different segments of the network.

$$f_*(\omega)|_{g^*(A?)} = 0.48 |w, r\rangle + 0.18 |w, r^\perp\rangle + 0.24 |w^\perp, r\rangle + 0.10 |w^\perp, r^\perp\rangle$$

We see that, in our model, roughly one half of the accepted papers had both qualities, but only 10% of them had none.

Lastly, we consider an example of forward inference. Following the recipe (4), we introduce a predicate  $E?: 2_T \otimes 2_S \rightarrow [0, 1]$  on the state  $\omega: 1 \rightarrow 2_T \otimes 2_S$ : it expresses the event that, while writing the paper, the scientist finds out that the main result contains a minor mistake and thus needs some revision.

$$(t, s) \mapsto \frac{2}{10} \quad (t, s^\perp) \mapsto \frac{4}{10} \quad (t^\perp, s) \mapsto \frac{3}{10} \quad (t^\perp, s^\perp) \mapsto \frac{6}{10}.$$

Differently from  $A?$ , this  $E?$  is a fuzzy predicate: a mistake gets more likely the less time and skill are available to the scientist. If this situation occurs, the scientist may still be able to produce on time a paper that gets accepted, but chances are lower: they decrease from 48% to 43%. This is expressed by the following inference.

$$(g \circ f)_*(\omega|_{E?}) = 0.43 |a\rangle + 0.57 |a^\perp\rangle$$

**Remark 3.1** We have modeled a Bayesian network as a graph in the Kleisli category  $\mathcal{Kl}(\mathcal{D})$ . This is inspired by the approach of Fong [8], except that he uses the Kleisli category  $\mathcal{Kl}(\mathcal{G})$  of the Girly monad (even though all his examples are discrete). Such graphs in  $\mathcal{Kl}(\mathcal{D})$  or  $\mathcal{Kl}(\mathcal{G})$  can be seen as symmetric monoidal functors from a PROP  $\mathcal{P}$ , generated by a signature with the nodes and edges of the network, to the Kleisli category. We recall that a PROP (**product** and **permutation** category [15]) is

a symmetric strict monoidal category with the natural numbers as objects and with monoidal product  $\oplus$  given by addition of numbers. Intuitively, PROPs generalise Lawvere theories from the cartesian to the linear setting; functors from  $\mathcal{P}$  as above are called the *models* of  $\mathcal{P}$ .

In our case, the model  $\mathcal{P} \rightarrow \mathcal{Kl}(\mathcal{D})$  sends  $\oplus$  to the monoidal product  $\otimes$  of  $\mathcal{Kl}(\mathcal{D})$ , and sends the number 1 to the object  $2 = 1 + 1$  in  $\mathcal{Kl}(\mathcal{D})$ .  $\mathcal{P}$  has pairing (copying)  $\multimap$ , but a crucial point is that these copiers are not natural — as can be checked easily in  $\mathcal{Kl}(\mathcal{D})$ . This implies that  $\mathcal{P}$  is not a Lawvere theory (cf. [3]), and there is no associated monad on **Set**.

This monad perspective comes up in the following way. A Bayesian network with set of nodes  $X$  can be seen as a coalgebra of the form:

$$X \xrightarrow{c} \mathcal{B}(X) \quad \text{where} \quad \mathcal{B}(X) = \coprod_{U \subseteq_{\text{fin}} X} [0, 1]^{2^{\#U}}$$

This coalgebra  $c$  sends a node  $N \in X$  to a pair  $c(N) = \langle c_1(N), c_2(N) \rangle$  where  $c_1(N) \subseteq_{\text{fin}} X$  is a finite set of predecessor nodes of  $N$ , and  $c_2(N): 2^n \rightarrow [0, 1]$  is the associated conditional probability table — where  $n = \#c_1(N) \in \mathbb{N}$  is the number predecessors. Since  $[0, 1] \cong \mathcal{D}(2)$ , this map  $c_2(N)$  is a Kleisli map  $2^n \rightarrow 2$  in  $\mathcal{Kl}(\mathcal{D})$ , as used in the above description of the paper-acceptance example.

It is not hard to see that the mapping  $X \mapsto \mathcal{B}(X)$  is a functor on **Set**, and comes with a unit map  $X \rightarrow \mathcal{B}(X)$ . But  $\mathcal{B}$  is not a monad, at least not in the expected obvious sense, precisely because the copiers  $\multimap$  are not natural.

## 4 Inference with continuous probability

Our abstract description of inference allows us to transfer the definitions from the discrete to the continuous approach simply by switching from the Kleisli category  $\mathcal{Kl}(\mathcal{D})$  of the distribution monad to the Kleisli category  $\mathcal{Kl}(\mathcal{G})$  of the Giry monad [9] on measurable spaces. We shall sketch an example where the function  $f$  in the inference situation (3) is the identity, but where we have multiple predicates  $p_i$  for successive learning. Hence there is no predicate/state transformation involved. We describe the essentials and refer to [5] for more information.

A state  $\omega: 1 \rightarrow X$  in the Kleisli category  $\mathcal{Kl}(\mathcal{G})$  is a probability measure  $\omega \in \mathcal{G}(X)$ , given by a function  $\omega: \Sigma_X \rightarrow [0, 1]$  that maps measurable subsets to probabilities. A predicate  $p: X \rightarrow 2$  in  $\mathcal{Kl}(\mathcal{G})$  is a measurable function  $p: X \rightarrow [0, 1]$  since  $\mathcal{G}(2) \cong [0, 1]$ . The validity  $\omega \models p$  in  $[0, 1]$  and conditional state  $\omega|_p$  in  $\mathcal{G}(X)$  are given by the following integration formulas.

$$\omega \models p := \int p \, d\omega \quad \text{and} \quad \omega|_p(M) := \frac{\int_M p \, d\omega}{\omega \models p}. \quad (11)$$

Often the state/probability measure  $\omega$  that we start from is given by a probability density function. This means that  $\omega$  is of the form  $\phi \models q$ , for some predicate  $q$ . In that case the conditional state  $\omega|_p = (\phi|_q)|_p$  is the same as the condition of the product predicate:  $\phi|_{q \cdot p}$  with pdf  $q \cdot p$ . This greatly simplifies the picture below.

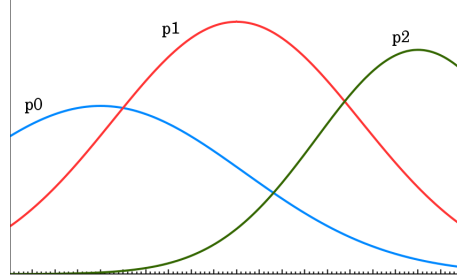
The inference example that we use is a continuous version of the archeological example described in [13]. The aim is to infer the date of a tomb at an archeological

site of which we already know that it is from the interval  $0 - 100$  AD. We are specifically looking to find three kinds of objects, labelled  $0, 1, 2$  of which we know the time of use more precisely. They are used to infer the age of the tomb. This knowledge is represented by three predicates  $p_0, p_1, p_2: [0, 100] \rightarrow [0, 1]$  given by the formulas:

$$p_0(x) = 0.6 \cdot e^{-(x-20)^2/2000}$$

$$p_1(x) = 0.9 \cdot e^{-(x-50)^2/1500}$$

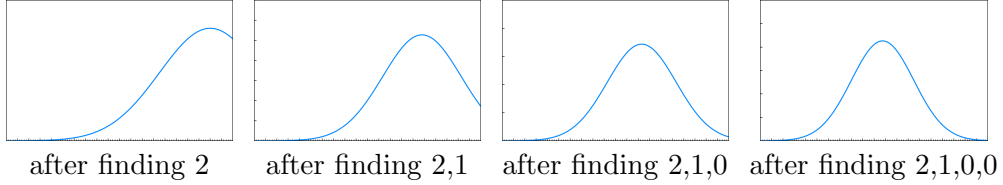
$$p_2(x) = 0.8 \cdot e^{-(x-90)^2/1000}$$



Our inference works as follows. We start from the uniform measure  $\omega = \phi|_q$  with pdf  $q(x) = \frac{1}{100}$  on  $[0, 100]$ , for the Lebesgue measure  $\phi$ . Its probability on the sub-interval  $[a, b] \subseteq [0, 100]$  is given by the integral:

$$\omega([a, b]) = \phi|_q([a, b]) = \int_a^b q \, d\phi = \int_a^b \frac{1}{100} \, d\phi = \frac{b-a}{100}.$$

We now successively observe objects  $i_1, \dots, i_n$ , for  $i_k = 0, 1, 2$ , and compute the conditional probability measure  $(\dots(\omega|_{p_{i_1}}) \dots)|_{p_{i_n}}$ . We can describe this measure via the product pdf  $q \cdot p_{i_1} \dots p_{i_n}$ , after normalisation. Below we sketch the shape of some of the resulting pdf's (ignoring normalisation), after finding certain objects successively.



These curves describe the inferred probability for the age of the tomb in the interval  $0 - 100$  AD.

## 5 Quantum inference

Our inference situations (3) and (4) can also be interpreted in the effectus of von Neumann algebras for quantum computation. Actually, one uses the opposite  $\mathbf{vNA}^{\text{op}}$  of the category  $\mathbf{vNA}$  of von Neumann algebras, with normal completely positive unital maps between them (see [5] for details). We have to take the opposite category because maps between von Neumann algebras should be understood as predicate transformers. Typical examples are the von Neumann algebras  $\mathcal{B}(\mathcal{H})$  of bounded operators on a Hilbert space  $\mathcal{H}$ . Below we use the matrix algebra  $M_2 = \mathcal{B}(\mathbb{C}^2)$  as special case.

For instance, the situation (3) translates into a diagram of maps in the category

**vNA** pointing in the other direction:

$$\mathbb{C} \xleftarrow{\omega} \mathcal{B} \xleftarrow{f} \mathcal{A} \xleftarrow{q} \mathbb{C}^2$$

The conditional state  $\omega|_{f^*(q)}: \mathcal{B} \rightarrow \mathbb{C}$  in backward inference is given by the general formula:

$$b \mapsto \frac{\omega(\sqrt{f(q)} \cdot b \cdot \sqrt{f(q)})}{\omega(f(q))}. \quad (12)$$

In this situation predicate transformation  $f^*(q) = f \circ q$  works in the opposite direction. The square-roots arise from the particular form of ‘assert’ map that is used for von Neumann algebras, see [5] for details. The predicate  $q: \mathbb{C}^2 \rightarrow \mathcal{A}$  is a positive unital map, and can thus be identified with an effect in  $\mathcal{A}$ , that is, with an element  $q \in \mathcal{A}$  satisfying  $0 \leq q \leq 1$ .

Bayesian inference in a quantum setting is a relatively new topic, see *e.g.* [14,6]. At this stage we only apply our general pattern from Definition 2.1 in a quantum setting. The illustration below repeats the disease-test example from Section 3 for the von Neumann algebra  $M_2$  of  $2 \times 2$  complex matrices. Our only ambition at this stage is to show how the quantum description extends the probabilistic one. Consider therefore the diagram:

$$\mathbb{C} \xleftarrow{\omega} M_2 \xleftarrow{s} M_2 \ni T? = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

These test (sensitivity) and state maps are given by:

$$s\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} \frac{9}{10}a + \frac{1}{10}d & 0 \\ 0 & \frac{1}{20}a + \frac{19}{20}d \end{pmatrix} \quad \text{and} \quad \omega\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \frac{1}{100}a + \frac{99}{100}d.$$

Predicate transformation yields:

$$s(T?) = s\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} \frac{9}{10} & 0 \\ 0 & \frac{1}{20} \end{pmatrix} \quad \text{and} \quad \omega(s(T?)) = \frac{117}{2000}.$$

The backward inferred state  $\omega|_{s^*(T?)}$  is according to (12):

$$\begin{aligned} \begin{pmatrix} a & b \\ c & d \end{pmatrix} &\mapsto \frac{2000}{117} \cdot \omega \left( \begin{pmatrix} \sqrt{9/10} & 0 \\ 0 & \sqrt{1/20} \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \sqrt{9/10} & 0 \\ 0 & \sqrt{1/20} \end{pmatrix} \right) \\ &= \frac{2000}{117} \cdot \omega \begin{pmatrix} 9/10 a & \sqrt{9/200} b \\ \sqrt{9/200} c & 1/20 d \end{pmatrix} \\ &= \frac{18}{117}a + \frac{99}{117}d. \end{aligned}$$

We see that the outcome is the same, up to some re-shuffling, as in the discrete probabilistic presentation in (8). But this situation allows much richer structure, for instance using as state  $\rho: M_2 \rightarrow \mathbb{C}$  the map  $\rho\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \frac{1}{2}(a - b - c + d)$ .

### Conclusions

This paper has clarified the role of states and predicates, and of state transformers and predicate transformers, in Bayesian inference. An abstract definition of for-



ward and backward inference has been given in the context of effectus theory, and interpreted and elaborated in several contexts and examples.

The generality of our approach allows for applications outside of the traditional probabilistic setting; the case of Von Neumann algebras is one such example which has been described here only in limited, probabilistic form. The power of the properly quantum approach (see also [14,6]) will be elaborated elsewhere.

The application to Bayesian networks also leaves room for interesting developments. As sketched in Remark 3.1, the interpretation of networks as arrows of  $\mathcal{Kl}(\mathcal{D})$  can be seen as part of a broader picture, that can be formulated in the language of PROPs and their models. We find particularly worthwhile trying to understand Bayesian inference, as introduced in the present paper, as a categorical transformation on models of a PROP: it should map one network into another one with the same topology, but different probability distributions.

### Acknowledgements

The authors acknowledge support from the European Research Council under the European Union’s Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n° 320571.

## References

- [1] Adams, R. and B. Jacobs, *A type theory for probabilistic and Bayesian reasoning* (2015), see [arxiv.org/abs/1511.09230](https://arxiv.org/abs/1511.09230).
- [2] Barber, D., “Bayesian Reasoning and Machine Learning,” Cambridge Univ. Press, 2012.
- [3] Bonchi, F., P. Sobocinski and F. Zanasi, *Lawvere categories as composed PROPs*, in: *Coalgebraic Methods in Computer Science (CMCS 2016)*, colocated with ETAPS 2016, 2016, to appear.
- [4] Borgström, J., A. Gordon, M. Greenberg, J. Margetson and J. V. Gael, *Measure transformer semantics for Bayesian machine learning*, Logical Methods in Comp. Sci. **9(3)** (2013), pp. 1–39.
- [5] Cho, K., B. Jacobs, A. Westerbaan and B. Westerbaan, *An introduction to effectus theory* (2015), see [arxiv.org/abs/1512.05813](https://arxiv.org/abs/1512.05813).
- [6] Coecke, B. and R. Spekkens, *Picturing classical and quantum Bayesian inference.*, Synthese **186** (2012), pp. 651–696.
- [7] Dijkstra, E. W., “A Discipline of Programming,” Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997, 1st edition.
- [8] Fong, B., “Causal Theories: A Categorical Perspective on Bayesian Networks,” Master’s thesis, Univ. of Oxford (2012), see [arxiv.org/abs/1301.6201](https://arxiv.org/abs/1301.6201).
- [9] Giry, M., *A categorical approach to probability theory*, in: B. Banaschewski, editor, *Categorical Aspects of Topology and Analysis*, number 915 in Lect. Notes Math. (1982), pp. 68–85.
- [10] Goodman, N. D., *The principles and practice of probabilistic programming*, in: *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL ’13 (2013), pp. 399–402.
- [11] Jacobs, B., *New directions in categorical logic, for classical, probabilistic and quantum logic*, Logical Methods in Comp. Sci. **11(3)** (2015), pp. 1–76.
- [12] Jacobs, B., *A recipe for state and effect triangles*, in: L. Moss and P. Sobocinski, editors, *Conference on Algebra and Coalgebra in Computer Science (CALCO 2015)*, LIPIcs **35** (2015), pp. 116–129.
- [13] Jacobs, B., B. Westerbaan and A. Westerbaan, *States of convex sets*, in: A. Pitts, editor, *Foundations of Software Science and Computation Structures*, number 9034 in Lect. Notes Comp. Sci. (2015), pp. 87–101.

- [14] Leifer, M. and R. Spekkens, *Towards a formulation of quantum theory as a causally neutral theory of Bayesian inference*, Phys. Rev. A **88(5)** (2013), p. 052130.
- [15] Mac Lane, S., *Categorical algebra*, B Am Math Soc **71** (1965), pp. 40–106.
- [16] Pearl, J., “Probabilistic Reasoning in Intelligent Systems,” Graduate Texts in Mathematics 118, Morgan Kaufmann, 1988.
- [17] Pearl, J., “Causality. Models, Reasoning, and Inference,” Cambridge Univ. Press, 2009, 2<sup>nd</sup> ed. edition.
- [18] Russell, S. and P. Norvig, “Artificial Intelligence. A Modern Approach,” Prentice Hall, 2003.
- [19] Ścibior, A., Z. Ghahramani and A. Gordon, *Practical probabilistic programming with monads*, in: *Proc. 2015 ACM SIGPLAN Symp. on Haskell* (2015), pp. 165–176.
- [20] Selinger, P., *A survey of graphical languages for monoidal categories*, Springer Lecture Notes in Physics **13** (2011), pp. 289–355, available at <http://arxiv.org/abs/0908.3347>.
- [21] Staton, S., H. Yang, C. Heunen, O. Kammar and F. Wood, *Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints*, in: *Logic in Computer Science (LICS 2016)*, 2016, to appear, available at <http://arxiv.org/abs/1601.04943>.

# Bitopology and four-valued logic

Tomáš Jakl<sup>1,2</sup>

*Department of Applied Mathematics  
MFF, Charles University  
Prague, Czech Republic*

*and*

*School of Computer Science  
University of Birmingham  
Birmingham, UK*

Achim Jung<sup>3</sup>

*School of Computer Science  
University of Birmingham  
Birmingham, UK*

Aleš Pultr<sup>1,4</sup>

*Department of Applied Mathematics  
MFF, Charles University  
Prague, Czech Republic*

---

## Abstract

Bilattices and d-frames are two different kinds of structures with a four-valued interpretation. Whereas d-frames were introduced with their topological semantics in mind, the theory of bilattices has a closer connection with logic. We consider a common generalisation of both structures and show that this not only still has a clear bitopological semantics, but that it also preserves most of the original bilattice logic. Moreover, we also obtain a new bitopological interpretation for the connectives of four-valued logic.

*Keywords:* Bilattices, d-frames, nd-frames, bitopological spaces, four-valued logic.

---

## 1 Introduction

In 1977, Nuel D. Belnap [5] gave a philosophical justification for distinguishing between two orders when studying information systems: the information order and the logical order. He also suggested that in addition to the classical logical values

---

<sup>1</sup> The work was supported by the grant SVV-2016-260332 and by the CE-ITI grant, GAČR P202/12/G061.

<sup>2</sup> Email: [jaklt@kam.mff.cuni.cz](mailto:jaklt@kam.mff.cuni.cz)

<sup>3</sup> Email: [A.Jung@cs.bham.ac.uk](mailto:A.Jung@cs.bham.ac.uk)

<sup>4</sup> Email: [pultr@kam.mff.cuni.cz](mailto:pultr@kam.mff.cuni.cz)

true and false (denoted  $tt$  and  $ff$ ) it would also be useful to have the values  $\top$  and  $\perp$  for information-wise maximum and minimum, corresponding to the situation when there is contradicting information and no information, respectively.

Belnap's insights and then Ginsberg's application of those ideas to inference systems [9] motivated Arieli and Avron to develop a formal logical system and to analyse its algebraic models – bilattices [3]. Bilattices proved to be also useful in other areas such as in logic programming [7], algebra [11], and abstract algebraic logic [14].

Later on Jung and Moshier studied bitopological spaces (or *bispaces* for short) to clarify the interplay of various topologies arising in domain theory. They discovered that bispaces actually also give a very natural semantics to a four-valued logic. The fact that the first topology can represent the observably true predicates and the second the observably false ones gives a good four-valued interpretation/reading and this is even more transparent in the algebraic duals of bitopological spaces, dubbed *d-frames* in [10].

Whereas d-frames were introduced with their topological semantics in mind, the theory of bilattices has a closer connection with logic. Because of this, one might not expect many similarities between both theories but the discovery of the so-called *twist-representation* of bilattices [6] shows that quite the opposite is the case.

In this paper, we are trying to tackle the obvious question of whether there is a reasonable generalisation of both theories that would give us some better insight into the similarities and differences between bilattices and d-frames. We claim that the answer to this question is yes. As a starting point we take d-frames and we will show that they can be very naturally extended into a new structure which we call *nd-frames*. It seems that this way we get the best from both worlds: We have a clear bitopological semantics while still preserving most of the original logic of bilattices. Moreover, in the nd-frame context the negation of four-valued logic has a new and clear bitopological realisation via interior operators.

This paper contributes to both the study of bilattices and the study of d-frames. For the former, it shows how to generalise bilattices to get four-valued structures where the components are not isomorphic. Contributions to d-frame theory are by giving an explanation of proof-theoretic negation and, moreover, extending this negation to the whole d-frame. By this we also show another connection between geometry (interior operators) and proof theory (cut rules). Moreover, nd-frames allow a finer distinction of bispaces as the class of their spectra is broader than the class of spectra of d-frames. Having a generalisation of both structures allows us to compare partial implication in d-frames with the implication of bilattices and to show that the former is much stronger than latter.

## 2 Preliminaries

Below we give very brief presentations of the two types of structures that this paper aims to combine, *bilattices* and *d-frames*. As will soon become clear, much of the underlying structure is *symmetric* with respect to a positive and a negative part; we will respect this when stating a definition or a proposition but will generally restrict proofs to one variant without further comment.

## 2.1 Bilattices

Bilattices are the algebraic manifestation of Belnap’s “useful four-valued logic”, [5]. One key feature of this logic is *paraconsistency*, [13], which means that it is *not* possible to derive an arbitrary proposition from a contradiction. Secondly, the logic is *truth-functional* in that every connective is characterised by its behaviour on the set of (four) truth values.

Traditionally, *bilattices* are presented as structures of the type  $(A; \wedge, \vee, \sqcap, \sqcup, \text{ff}, \text{tt}, \perp, \top, \neg, \supset)$  satisfying a list of axioms. However, a decomposition theorem can be shown for them (see [4,14,6]) and it is more straightforward to approach the subject from the characterisation that results from it.<sup>5</sup>

Let  $H = (H; \wedge, \vee, 1, 0, \rightarrow)$  be a Heyting algebra. On  $H \times H$  one defines the bilattice operations by setting, for  $\alpha = (\alpha_+, \alpha_-)$ ,  $\beta = (\beta_+, \beta_-) \in H \times H$ :<sup>6</sup>

$$\begin{aligned} \alpha \vee \beta &\stackrel{\text{def}}{=} (\alpha_+ \vee \beta_+, \alpha_- \wedge \beta_-), & \alpha \wedge \beta &\stackrel{\text{def}}{=} (\alpha_+ \wedge \beta_+, \alpha_- \vee \beta_-), \\ \alpha \sqcup \beta &\stackrel{\text{def}}{=} (\alpha_+ \vee \beta_+, \alpha_- \vee \beta_-), & \alpha \sqcap \beta &\stackrel{\text{def}}{=} (\alpha_+ \wedge \beta_+, \alpha_- \wedge \beta_-), \\ \text{ff} &\stackrel{\text{def}}{=} (0, 1), & \text{tt} &\stackrel{\text{def}}{=} (1, 0), & \perp &\stackrel{\text{def}}{=} (0, 0), & \top &\stackrel{\text{def}}{=} (1, 1). \end{aligned}$$

The two final operations deserve to be highlighted: *Negation*  $\neg$  is defined purely by the exchange of components:

$$\neg \alpha \stackrel{\text{def}}{=} (\alpha_-, \alpha_+)$$

and without reference to the internal logical structure of the component Heyting algebra. *Weak implication*  $\supset$  is the only *non-symmetric* operation in the signature and it is in fact a remarkable feature of bilattice logic that it can be given in the following way at all:

$$\alpha \supset \beta \stackrel{\text{def}}{=} (\alpha_+ \rightarrow \beta_+, \alpha_+ \wedge \beta_-)$$

The set  $H \times H$  together with the four constants and six operations defined above is called the *twist-construction* over  $H$  and denoted by  $H^{\boxtimes}$ . As we said above, the characterisation theorem states that, up to isomorphism, every bilattice arises in this way.

Notice that the “logical” reduct  $(H \times H; \wedge, \vee, \text{ff}, \text{tt})$  and the “informational” reduct  $(H \times H; \sqcap, \sqcup, \perp, \top)$  are automatically bounded distributive lattices. The associated orders,  $\leq$  and  $\sqsubseteq$ , however, are not the same; they may be (loosely) said to be “at 90° to each other”, and this helps to explain that negation  $\neg$  is antitone w.r.t. the logical order and monotone w.r.t. the informational one.

## 2.2 D-frames

The motivation for d-frames comes from semantics, in particular, from the observation that “domains” (in the sense of Scott) carry two topologies which are loosely

<sup>5</sup> The decomposition theorem is surprisingly robust; very little of the structure of bilattices is required. While this is an intriguing aspect of the theory, it has also led to a proliferation of terminology, not all of which is universally accepted. Our choice of “bilattice” for the purposes this paper is really just for brevity and simplicity as we will have no need to consider any variations in the axiomatisation.

<sup>6</sup> Note the overloading of  $\wedge$  and  $\vee$  as operations on both the Heyting algebra and the bilattice. We hope that the context will always make clear what we are referring to.

connected and in some sense complementary to each other, the *Scott-topology* and the *weak lower topology*, [2,8]. Smyth, [15], proposed to interpret open sets as propositions of an “observational logic”, and Abramsky fully developed this programme in his celebrated “Domain Theory in Logical Form”, [1], but both these works focus entirely on the Scott-topology which begged the question what the logical status of the other topology might be. Taking a step back from domain theory, [10] began the exploration of bitopological spaces under Smyth’s interpretation. D-frames are the result of this investigation.

A *d-frame*<sup>7</sup> is a structure  $\mathcal{L} = (L_+ \times L_-; \text{con}, \text{tot})$  where  $(L_+; \vee, \wedge, 0, 1)$  and  $(L_-; \vee, \wedge, 0, 1)$  are frames<sup>8</sup> and the *consistency*  $\text{con} \subseteq L_+ \times L_-$  and *totality*  $\text{tot} \subseteq L_+ \times L_-$  predicates satisfy the following axioms, for all  $\alpha, \beta \in L_+ \times L_-$ :

$$\begin{aligned}
 (\text{con}-\downarrow) \quad & \alpha \in \text{con} \text{ and } \beta \sqsubseteq \alpha \implies \beta \in \text{con}, \\
 (\text{tot}-\uparrow) \quad & \alpha \in \text{tot} \text{ and } \beta \sqsupseteq \alpha \implies \beta \in \text{tot}, \\
 (\text{con}-\wedge, \vee) \quad & \alpha, \beta \in \text{con} \implies \alpha \vee \beta \in \text{con} \text{ and } \alpha \wedge \beta \in \text{con}, \\
 (\text{tot}-\wedge, \vee) \quad & \alpha, \beta \in \text{tot} \implies \alpha \vee \beta \in \text{tot} \text{ and } \alpha \wedge \beta \in \text{tot}, \\
 (\text{con}, \text{tot}-\text{tt}, \text{ff}) \quad & \text{tt} \in \text{con} \text{ and } \text{tt} \in \text{tot}, \quad \text{ff} \in \text{con} \text{ and } \text{ff} \in \text{tot}, \\
 (\text{con}-\text{tot}) \quad & \alpha \in \text{con}, \beta \in \text{tot} \text{ and } (\alpha \sqcup \beta = \alpha \wedge \beta \text{ or } \alpha \sqcup \beta = \alpha \vee \beta) \implies \alpha \sqsubseteq \beta \\
 (\text{con}-\sqcup^\uparrow) \quad & A \subseteq \text{con} \text{ and } A \text{ is } \sqsubseteq\text{-directed} \implies \sqcup^\uparrow A \in \text{con}.
 \end{aligned}$$

where  $\wedge, \vee, \sqcap, \sqcup, \text{ff}, \text{tt}, \perp, \top$  and the induced logical order  $\leq$  and information order  $\sqsubseteq$  are defined the same way as in bilattices. In fact, the similarity with bilattices (presented as twist structures) is obvious and it may therefore be helpful to highlight the *differences*:

- In d-frames, the two component lattices may be different, in bilattices they are identical;
- (consequently) it is not possible to define negation or weak implication on d-frames in the same way as it is done for bilattices;
- frames are *complete* Heyting algebras (but frame homomorphisms may not preserve Heyting implication);
- the two predicates  $\text{con}$  and  $\text{tot}$  are *relational*, not *algebraic* structure.

These differences are also apparent in the definition of *d-frame homomorphism* which we take to be a pair of frame homomorphisms  $h_+ : L_+ \rightarrow M_+, h_- : L_- \rightarrow M_-$  such that  $h_+ \times h_-[\text{con}_{\mathcal{L}}] \subseteq \text{con}_{\mathcal{M}}$  and  $h_+ \times h_-[\text{tot}_{\mathcal{L}}] \subseteq \text{tot}_{\mathcal{M}}$ . We denote the category of d-frames and d-frame homomorphisms by **d-Frm**.

As we said before, d-frames arose from consideration of bitopological spaces and indeed, it is straightforward to adapt the open-set functor from spaces to frames to one from the category **biTop** of bispaces to **d-Frm**. To this end, we set  $\Omega_d(X) = (\tau_+, \tau_-; \text{tot}_X, \text{con}_X)$  for a bispace  $(X; \tau_+, \tau_-)$  where, for  $U \in \tau_+$  and  $V \in \tau_-$ ,

$$(U, V) \in \text{con}_X \stackrel{\text{def}}{=} U \cap V = \emptyset \quad \text{and} \quad (U, V) \in \text{tot}_X \stackrel{\text{def}}{=} U \cup V = X.$$

<sup>7</sup> This definition of d-frames agrees with the definition of reasonable d-frames in [10].

<sup>8</sup> *Frames* are complete lattices satisfying the equation:  $b \wedge (\bigvee_i a_i) = \bigvee_i (b \wedge a_i)$ . Frame homomorphisms are monotone maps distributing over all joins and finite meets. For more information see [12].

**Example 2.1** The dual of the one-point bispaces  $\mathbf{1} = (\{*\}; \tau, \tau)$  has exactly four elements:  $\perp = (\emptyset, \emptyset)$ ,  $\text{ff} = (\emptyset, \{*\})$ ,  $\text{tt} = (\{*\}, \emptyset)$ , and  $\top = (\{*\}, \{*\})$ . These are the truth values of bilattice logic and in that context the structure is usually denoted  $\mathcal{FOUR}$ . What we are saying here is that  $\mathcal{FOUR}$  is also a canonical d-frame with component frames  $L_+ = L_- = \mathbf{2} = \{0 < 1\}$ .

### 3 Nd-frames

We saw in the Preliminaries that, in order to define negation and implication for bilattices represented as twist-structures, we heavily use the fact that the carrier is the product of a Heyting algebra with itself. Therefore, we can freely send elements from one component of the product to the other.

Similarly to bilattices, d-frames are also formed of two components but those do not have to be the same and it seems that there are no natural order-preserving mappings between them (as required by the definition of  $\neg$  and  $\supset$ ). For example, taking pseudocomplements<sup>9</sup> is antitone and it could be used to define an operation sometimes called *conflation*, but this is known to be different from negation.

However, looking at the semantic counterparts of d-frames, i.e. bitopological spaces, suggests that we have very natural candidates for maps between both frames of open sets. Let  $(X; \tau_+, \tau_-)$  be a bispaces; then assigning for every  $\tau_+$ -open (or  $\tau_-$ -open) set its interior with respect to the other topology is a monotone map. Moreover, it also distributes over intersections. Let us denote those maps by  $m: \tau_+ \rightarrow \tau_-$  and  $p: \tau_- \rightarrow \tau_+$ ; to wit:

$$m: U \in \tau_+ \mapsto U^{\circ\tau_-} \in \tau_- \quad \text{and} \quad p: V \in \tau_- \mapsto V^{\circ\tau_+} \in \tau_+.$$

When translated to the language of d-frames, one can postulate the existence of maps  $m: L_+ \rightarrow L_-$  and  $p: L_- \rightarrow L_+$  satisfying the following axioms:

- (pm-1)  $m(a \wedge b) = m(a) \wedge m(b)$ ,  $p(a \wedge b) = p(a) \wedge p(b)$ ,
- (pm-2)  $m(1) = 1$ ,  $p(1) = 1$ ,
- (pm-3)  $m(0) = 0$ ,  $p(0) = 0$ ,
- (pm-4)  $p \circ m \leq \text{id}$ ,  $m \circ p \leq \text{id}$ .

Also, the intuition of  $p$  and  $m$  being the interiors with respect to the other topology justifies the following axioms involving **con** and **tot**:

$$\begin{array}{c} \frac{(a \wedge b, c) \in \text{con}}{(a, m(b) \wedge c) \in \text{con}} \text{ (con-}m\text{)} \quad \frac{(a, b \wedge c) \in \text{con}}{(a \wedge p(b), c) \in \text{con}} \text{ (con-}p\text{)} \\[10pt] \frac{(a, m(b) \vee c) \in \text{tot}}{(a \vee b, c) \in \text{tot}} \text{ (tot-}m\text{)} \quad \frac{(a \vee p(b), c) \in \text{tot}}{(a, b \vee c) \in \text{tot}} \text{ (tot-}p\text{)} \end{array}$$

**Definition 3.1**  $(L_+ \times L_-; \text{con}, \text{tot}; p, m)$  is an *nd-frame* if  $(L_+, L_-; \text{con}, \text{tot})$  is a d-frame and all axioms for  $(p, m)$  mentioned above, i.e. (pm-1), (pm-2), (pm-3), (pm-4), (con- $m$ ), (con- $p$ ), (tot- $m$ ) and (tot- $p$ ), are satisfied.

<sup>9</sup> See Section 6 for the definition.

Let us recall a known fact about continuous maps:

**Lemma 3.2** *Let  $f: X \rightarrow Y$  be a continuous map and let  $M \subseteq Y$ , then  $f^{-1}[M^\circ] \subseteq (f^{-1}[M])^\circ$ .*

This motivates the following definition:

**Definition 3.3** A d-frame homomorphism  $h: \mathcal{L} \rightarrow \mathcal{M}$  between two nd-frames is an *nd-frame homomorphism* if  $h_+ \circ p \leq p \circ h_-$  and  $h_- \circ m \leq m \circ h_+$ .

Since the component maps are monotone, it follows that nd-frame homomorphisms are closed under composition. We denote the resulting category with **nd-Frm**.

**Example 3.4** We have seen before that the bilattice  $\mathcal{FOUR}$  may alternately be viewed as a d-frame. The axioms (pm-1) – (pm-4) ensure that the identity function is the unique choice for both  $p$  and  $m$  so that  $\mathcal{FOUR} = \mathbf{2} \times \mathbf{2}$  also qualifies as an nd-frame. Whether we view it as a bilattice, d-frame, or nd-frame, we always denote it with  $\mathcal{FOUR}$ .

The theory of d-frames works best when the two topologies complement each other, in the sense of the closed sets of one approximating the opens of the other. Examples of this are given by the Scott-topology and the weak lower topology considered in domain theory. If this is not the case, then the two relations **con** and **tot** tend to be trivial, by which we mean that  $(a, b) \in \mathbf{con}$  iff one of two elements equals 0, and  $(a, b) \in \mathbf{tot}$  iff one of them equals 1. For the new structure  $m$  and  $p$ , the situation is exactly reversed. To see this, consider the following two bispaces:

- (i)  $X_1 = (\{a, b\}, \tau, \tau)$  where the only non-trivial open of  $\tau$  is  $\{a\}$ .
- (ii)  $X_2 = (\{aa, ab, ba, bb\}, \tau_+, \tau_-)$  where the only non-trivial open of  $\tau_+$  is  $\{aa, ab\}$  and that of  $\tau_-$  is  $\{aa, ba\}$ .

In both cases, **con** and **tot** are trivial, which means that  $\Omega_d(X_1)$  and  $\Omega_d(X_2)$  are isomorphic. On the other hand, the interior operators on  $X_1$  are the identity whereas on  $X_2$  they are trivial. Both bispaces will turn out to be nd-sober, whereas only  $X_2$  is d-sober.

## 4 Logic of nd-frames

We introduced  $p$  and  $m$  to be able to define negation and implication for d-frames similarly to how they are defined for bilattices in their twist-structures representation. Let  $(L_+ \times L_-; \mathbf{con}, \mathbf{tot}; p, m)$  be an nd-frame and define

$$\neg\varphi \stackrel{\text{def}}{=} (p(\varphi_-), m(\varphi_+)) \quad \text{and} \quad \varphi \supset \psi \stackrel{\text{def}}{=} (\varphi_+ \rightarrow \psi_+, m(\varphi_+) \wedge \psi_-).$$

Now any nd-frame  $\mathcal{L}$  gives rise to the same signature as bilattices:  $(L_+ \times L_-; \wedge, \vee, \sqcap, \sqcup, \mathbf{ff}, \mathbf{tt}, \perp, \top, \neg, \supset)$ . Let **Ln** be the language of bilattices and let **Fm(Ln)** be the term algebra of **Ln** (generated by countably many variables). Valuations are the **Ln**-homomorphisms  $\mathbf{Fm}(\mathbf{Ln}) \rightarrow \mathcal{L}$ . We can define (algebraic) semantic validity the same way as it was defined for bilattices [14]; we say  $\varphi$  *holds in* (or, *is*



valid in)  $\mathcal{L}$ , and write  $\mathcal{L} \models \varphi$  iff

$$v(\varphi) = v(\varphi \supset \varphi) \text{ for all valuations } v: \mathbf{Fm}(\mathbf{Ln}) \rightarrow \mathcal{L},$$

Before we get to the axioms let us first prove the following lemma.

**Lemma 4.1** *The following holds in all nd-frames  $\mathcal{L}$ :*

- (L1)  $\mathcal{L} \models \varphi$  iff  $v(\varphi) \sqsupseteq \mathbf{tt}$  (or equivalently:  $v(\varphi)_+ = 1$ ) for all valuations  $v$  into  $\mathcal{L}$ ;
- (L2)  $\mathcal{L} \models \varphi \supset \psi$  iff  $v(\varphi)_+ \leq v(\psi)_+$  for all valuations  $v$  into  $\mathcal{L}$ ; and
- (L3)  $\mathcal{L} \models \varphi \equiv \psi$  iff  $v(\varphi)_+ = v(\psi)_+$  for all valuations  $v$  into  $\mathcal{L}$ ,  
where  $\varphi \equiv \psi$  is a shorthand for  $(\alpha \supset \beta) \wedge (\beta \supset \alpha)$ .

**Proof.**

- (L1) Right-to-left implication: If  $v(\varphi) = (1, a)$  then  $v(\varphi \supset \varphi) = v(\varphi) \supset v(\varphi) = (1 \rightarrow 1, m(1) \wedge a) = (1, a) = v(\varphi)$ . Reverse direction:  $v(\varphi) = v(\varphi \supset \varphi)$  implies that the positive parts are equal and therefore we have  $v(\varphi)_+ = v(\varphi \supset \varphi)_+ = 1$  and  $v(\varphi) \sqsupseteq \mathbf{tt}$ .
- (L2) From (L1) we know that  $\mathcal{L} \models \varphi \supset \psi$  iff  $v(\varphi \supset \psi)_+ = v(\varphi)_+ \rightarrow v(\psi)_+$  is equal to 1 for all valuations  $v$ , and this is true if and only if  $v(\varphi)_+ \leq v(\psi)_+$ .
- (L3) Follows from (L2) and from the fact that  $\mathcal{L} \models v(\varphi \wedge \psi) \sqsupseteq \mathbf{tt}$  iff  $\mathcal{L} \models v(\varphi) \sqsupseteq \mathbf{tt}$  and  $\mathcal{L} \models v(\psi) \sqsupseteq \mathbf{tt}$ . Then,  $\mathcal{L} \models v(\varphi \equiv \psi)$  iff  $\mathcal{L} \models \varphi \supset \psi$  and  $\mathcal{L} \models \psi \supset \varphi$  iff  $v(\varphi)_+ \leq v(\psi)_+$  and  $v(\psi)_+ \leq v(\varphi)_+$  for all valuations  $v$ .

□

Arieli and Avron, [3], gave a Hilbert-style axiomatisation of a four-valued logic which is sound and complete with respect to bilattices. Here we show that a large part of their logic is still valid in nd-frames.

**Theorem 4.2** *The following axioms of four-valued logic are valid in any nd-frame: (Weak implication)*

$$\begin{aligned} (\supset 1) \quad & \varphi \supset (\psi \supset \varphi) \\ (\supset 2) \quad & (\varphi \supset (\psi \supset \gamma)) \supset ((\varphi \supset \psi) \supset (\varphi \supset \gamma)) \\ (\neg\neg R) \quad & \neg\neg\varphi \supset \varphi \end{aligned} \tag{\star A}$$

*(Logical conjunction and disjunction)*

$$\begin{aligned} (\wedge \supset) \quad & (\varphi \wedge \psi) \supset \varphi \text{ and } (\varphi \wedge \psi) \supset \psi \\ (\supset \wedge) \quad & \varphi \supset (\psi \supset (\varphi \wedge \psi)) \\ (\supset \mathbf{tt}) \quad & \varphi \supset \mathbf{tt} \\ (\supset \vee) \quad & \varphi \supset (\varphi \vee \psi) \text{ and } \psi \supset (\varphi \vee \psi) \\ (\vee \supset) \quad & (\varphi \supset \gamma) \supset ((\psi \supset \gamma) \supset ((\varphi \vee \psi) \supset \gamma)) \\ (\supset \mathbf{ff}) \quad & \mathbf{ff} \supset \varphi \end{aligned}$$

*(Informational conjunction and disjunction)*

$$\begin{aligned} (\sqcap \supset) \quad & (\varphi \sqcap \psi) \supset \varphi \text{ and } (\varphi \sqcap \psi) \supset \psi \\ (\supset \sqcap) \quad & \varphi \supset (\psi \supset (\varphi \sqcap \psi)) \end{aligned}$$

$$\begin{aligned}
 (\supset \top) \quad & \varphi \supset \top \\
 (\supset \sqcup) \quad & \varphi \supset (\varphi \sqcup \psi) \text{ and } \psi \supset (\varphi \sqcup \psi) \\
 (\sqcup \supset) \quad & (\varphi \supset \gamma) \supset ((\psi \supset \gamma) \supset ((\varphi \sqcup \psi) \supset \gamma)) \\
 (\supset \perp) \quad & \perp \supset \varphi
 \end{aligned}$$

(Negation)

$$\begin{aligned}
 (\neg \wedge L) \quad & \neg(\varphi \wedge \psi) \subset \neg\varphi \vee \neg\psi & (\star B) \\
 (\neg \vee) \quad & \neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi \\
 (\neg \sqcap) \quad & \neg(\varphi \sqcap \psi) \equiv \neg\varphi \sqcap \neg\psi \\
 (\neg \sqcup L) \quad & \neg(\varphi \sqcup \psi) \subset \neg\varphi \sqcup \neg\psi & (\star B) \\
 (\neg \supset R) \quad & \neg(\varphi \supset \psi) \supset \varphi \wedge \neg\psi & (\star A)
 \end{aligned}$$

Furthermore, the rule of Modus Ponens is sound:

$$(MP) \quad \varphi, (\varphi \supset \psi) \vdash \psi$$

**Proof.** The axioms  $(\supset 1)$ ,  $(\supset 2)$ , all the logical conjunction and disjunction axioms and the informational conjunction and disjunction axioms hold for the same reason. We know from Lemma 4.1 (L1) that only the first coordinate determines their validity. Moreover, those axioms do not contain negation and so they hold simply because, when projected to the first coordinate, they hold in all Heyting algebras (and therefore also in frames).

From (L2) we know that  $(\neg \wedge L)$  is equivalent to  $p(v(\varphi)_- \vee v(\psi)_-) \geq p(v(\varphi)_-) \vee p(v(\psi)_-)$  for all valuations  $v$  which is true since  $p$  is monotone. The same argument applies for  $(\neg \sqcup L)$ . From (L3) we know that  $(\neg \vee)$  and  $(\neg \sqcap)$  are equivalent to  $p(v(\varphi)_- \wedge v(\psi)_-) = p(v(\varphi)_-) \wedge p(v(\psi)_-)$  for all valuations  $v$  and this is true simply because  $p$  preserves finite infima.

(L2) implies that  $(\neg \supset R)$  is equivalent to  $v(\neg(\varphi \supset \psi))_+ \leq v(\varphi \wedge \neg\psi)_+$  by expanding the definitions we get  $(\neg v(\varphi \supset \psi))_+ = p(v(\varphi \supset \psi)_-) = p(m(v(\varphi)_+) \wedge v(\psi)_-) = p(m(v(\varphi)_+)) \wedge p(v(\psi)_-)$  which, by  $p \circ m \leq \text{id}$ , is less or equal to  $v(\varphi)_+ \wedge p(v(\psi)_-) = v(\varphi \wedge \neg\psi)_+$  as we wanted.

For Modus Ponens, no matter if we interpret comma as  $\wedge$  or as  $\sqcap$  we get the requirement that  $v(\varphi)_+ \wedge (v(\varphi)_+ \rightarrow v(\psi)_+) = 1$  should imply  $v(\psi)_+ = 1$  which is again true for all Heyting algebras.  $\square$

**Remark 4.3** The axioms denoted by  $(\star A)$  or  $(\star B)$  are the only axioms that differ from the original axioms of bilattices because they are expressed as implications, whereas the original axioms are equivalences. Requiring equivalence instead of implication in the axioms marked by  $(\star A)$  is equivalent to requiring that  $p \circ m = \text{id}$ <sup>10</sup> and requiring equivalences for axioms marked by  $(\star B)$  is the same as requiring that  $p$  preserves finite suprema. Also, in some presentations of bilattices, [3], the following axiom, called Peirce's law, is added

$$(\supset 3) \quad ((\varphi \supset \psi) \supset \varphi) \supset \varphi$$

Assuming this to hold is equivalent to assuming that  $L_+$  is a Boolean frame.

<sup>10</sup>Notice that assuming  $p \circ m = \text{id}$  implies that  $m$  is a one-one frame homomorphism and  $p$  is its right adjoint.

#### 4.1 Implications and a cut rule

One of the nice properties of d-frames is that one can restrict one's attention to the set  $\mathbf{con}$  of consistent predicates without losing any expressivity, see [10, Proposition 7.4]. We can think of this structure as a semantics for predicates *without contradictions*. Moreover, there is a binary relation between the elements of  $\mathbf{con}$  which is in many ways similar to a consequence relation (see Section 7 of [10]). Define, for all  $\alpha, \beta \in \mathbf{con}$ ,

$$\alpha \prec \beta \stackrel{\text{def}}{=} (\beta_+, \alpha_-) \in \mathbf{tot}.$$

Given the many similarities between d-frames and bilattices one wonders what exactly the relationship between  $\prec$  and  $\supset$  is. We would like to suggest that the right place to answer this question is within nd-frames as they generalise both notions. Indeed, we will see below that  $\prec$  is, from this perspective, much stronger than  $\supset$ .

**Theorem 4.4** *Let  $\mathcal{L}$  be an nd-frame and let  $\alpha, \beta \in \mathbf{con}$  such that  $\alpha \prec \beta$ . Then the following also hold*

- (i)  $\alpha' \supset \beta' \sqsupseteq \mathbf{tt}$  for all  $\alpha', \beta' \in \mathbf{con}$  with  $\alpha' \sqsupseteq \alpha$  and  $\beta' \sqsupseteq \beta$ ;
- (ii)  $\neg\beta' \supset \neg\alpha' \sqsupseteq \mathbf{tt}$  for all  $\alpha', \beta' \in \mathbf{con}$  with  $\alpha' \sqsupseteq \alpha$  and  $\beta' \sqsupseteq \beta$ ;

**Remark:** The logical conjunction of  $\alpha \supset \beta$  with  $\neg\beta \supset \neg\alpha$  is called *strong implication* in the bilattice logic literature.

**Proof.** We know from [10, Proposition 7.1(4)] that  $\alpha \prec \beta$  implies  $\alpha' \prec \beta'$  whenever  $\alpha' \sqsupseteq \alpha$  and  $\beta' \sqsupseteq \beta$  in  $\mathbf{con}$ , so it suffices to show the statements for  $\alpha$  and  $\beta$ : By Lemma 4.1 (L2),  $(\alpha \supset \beta) \sqsupseteq \mathbf{tt}$  iff  $\alpha_+ \leq \beta_+$  and this follows, by (con-tot), from  $(\beta_+, \alpha_-) \in \mathbf{tot}$  and  $\alpha \in \mathbf{con}$ . Similarly, since  $(\neg\beta \supset \neg\alpha) = (p(\beta_-) \rightarrow p(\alpha_-), mp(\beta_-) \wedge m(\alpha_+))$ , then  $(\neg\beta \supset \neg\alpha) \sqsupseteq \mathbf{tt}$  iff  $p(\beta_-) \rightarrow p(\alpha_-) = 1$  which is equivalent to  $p(\beta_-) \leq p(\alpha_-)$ . However, we know that  $(\beta_+, \alpha_-) \in \mathbf{tot}$  and  $\beta \in \mathbf{con}$ , therefore, by (con-tot),  $\beta_- \leq \alpha_-$  and, since  $p$  is monotone, also  $p(\beta_-) \leq p(\alpha_-)$ .  $\square$

The (tot- $m$ ) and (tot- $p$ ) axioms give us immediately the following two rules combining strong implication and negation:

$$\frac{\alpha \prec \neg\beta \vee \gamma}{\alpha \wedge \beta \prec \gamma} \quad \text{and} \quad \frac{\alpha \wedge \neg\beta \prec \gamma}{\alpha \prec \beta \vee \gamma}$$

and from these the following cut rule follows by the transitivity of  $\prec$  [10, Proposition 7.1 (3)]:

$$\frac{\alpha \prec \neg\neg\beta \vee \gamma \quad \gamma \wedge \neg\neg\alpha' \prec \beta'}{\alpha \wedge \alpha' \prec \beta \vee \beta'}$$

## 5 Stone duality for nd-frames

### 5.1 Spectra of nd-frames

In this section we define a spectrum functor  $\Sigma: \mathbf{nd-Frm} \rightarrow \mathbf{biTop}$  by extending the definition of the spectrum functor for d-frames  $\Sigma_d: \mathbf{d-Frm} \rightarrow \mathbf{biTop}$  as presented

in [10]. Let  $\mathcal{L} = (L_+ \times L_-; \mathbf{con}, \mathbf{tot}; p, m)$  be an nd-frame. Define  $\Sigma(\mathcal{L})$  to be the bispace  $(\Sigma(\mathcal{L}); \Phi_+[L_+], \Phi_-[L_-])$  where the underlying set  $\Sigma(\mathcal{L})$  is the set of *nd-points*, that is, the pairs  $(F_+, F_-)$  where  $F_+$  and  $F_-$  are complete prime filters of  $L_+$  and  $L_-$ , respectively, such that, for all  $\alpha \in L_+ \times L_-$ ,

$$\begin{aligned} (\mathbf{dp}_{\mathbf{con}}) \quad & \alpha \in \mathbf{con} \implies \alpha_+ \notin F_+ \text{ or } \alpha_- \notin F_-; \\ (\mathbf{dp}_{\mathbf{tot}}) \quad & \alpha \in \mathbf{tot} \implies \alpha_+ \in F_+ \text{ or } \alpha_- \in F_-; \\ (\mathbf{dp}_p) \quad & p(\alpha_-) \in F_+ \implies \alpha_- \in F_-; \\ (\mathbf{dp}_m) \quad & m(\alpha_+) \in F_- \implies \alpha_+ \in F_+. \end{aligned}$$

Equivalently, we can define the underlying set of  $\Sigma(\mathcal{L})$  to be the set of all nd-frame homomorphisms from  $\mathcal{L}$  to  $\mathcal{FOUR}$ . The topologies of  $\Sigma(\mathcal{L})$  are defined the same way as for spectra of d-frames. That is,  $\Phi_+[L_+] = \{\Phi_+(a) : a \in L_+\}$  and  $\Phi_-[L_-] = \{\Phi_-(b) : b \in L_-\}$  where

$$\Phi_+(a) = \{(F_+, F_-) \mid a \in F_+\} \quad \text{and} \quad \Phi_-(b) = \{(F_+, F_-) \mid b \in F_-\}.$$

Also, similarly to the d-frame spectrum functor, for every nd-frame homomorphism  $h: \mathcal{L} \rightarrow \mathcal{M}$ , set  $\Sigma(h): \Sigma(\mathcal{M}) \rightarrow \Sigma(\mathcal{L})$  to be the map

$$\Sigma(h): (F_+, F_-) \longmapsto (h_+^{-1}[F_+], h_-^{-1}[F_-]).$$

**Proposition 5.1**  $\Sigma$  is a contravariant functor from **nd-Frm** to **biTop**.

**Proof.**  $\Sigma$  is well defined on objects for the same reason as the corresponding functor for d-frames [10]. When we think of the nd-points of an nd-frame  $\mathcal{L}$  as nd-frame homomorphisms  $\mathcal{L} \rightarrow \mathcal{FOUR}$  we see that  $\Sigma$  is also well defined on **nd-Frm** morphisms simply because nd-frame homomorphisms are closed under composition.  $\square$

## 5.2 Nd-frames from bispaces

Let  $X = (X; \tau_+, \tau_-)$  be a bispace. Set  $\Omega(X) = (\tau_+, \tau_-; \mathbf{con}_X, \mathbf{tot}_X; p_X, m_X)$  where  $\mathbf{con}_X$  and  $\mathbf{tot}_X$  are as before and

$$m_X: U_+ \longmapsto U_+^{\circ\tau_-} \quad \text{and} \quad p_X: U_- \longmapsto U_-^{\circ\tau_+}.$$

Again,  $\Omega$  acts on morphisms the same way as the d-frames analogue does, i.e. for a bicontinuous map  $f: X \rightarrow Y$  set  $\Omega(f): \Omega Y \rightarrow \Omega X$  to be the map

$$\Omega(f): (U_+, U_-) \longmapsto (f^{-1}[U_+], f^{-1}[U_-]).$$

**Proposition 5.2**  $\Omega$  is a contravariant functor from **biTop** to **nd-Frm**.

**Proof.**  $\Omega$  is clearly well defined on objects. From the duality for d-frames, we know that the  $\Omega$ -image of a bicontinuous map is a d-frame homomorphism. The fact that it is also an nd-frame homomorphism, that is  $f(\neg\alpha) \sqsubseteq \neg(f\alpha)$  for all  $\alpha$ , follows directly from Lemma 3.2.  $\square$

### 5.3 Sobriety, spatiality and the adjunction

We say that a bispace  $X$  is *nd-sober* if there exists an nd-frame  $\mathcal{L}$  such that  $X \cong \Sigma(\mathcal{L})$ . We also have the usual embedding into the *sobrification* of a space (the *unit* of adjunction)  $\eta_X: X \rightarrow \Sigma\Omega(X)$  defined as  $x \mapsto (\mathcal{U}_+(x), \mathcal{U}_-(x))$  where  $\mathcal{U}_+(x)$  and  $\mathcal{U}_-(x)$  are the neighbourhood filters in  $\tau_+$  and  $\tau_-$ , respectively. For the same reason as in the case of d-frames,  $\eta_X$  is natural in  $X$ .

**Theorem 5.3** *For a bitopological space  $X$ , the following are equivalent:*

- (i)  $X$  is nd-sober;
- (ii)  $X$  is bihomeomorphic to  $\Sigma\Omega(X)$ ;
- (iii) The unit map  $\eta_X$  is a bihomeomorphism;
- (iv) The unit map  $\eta_X$  is a bijection.

The reader may now check that the two examples we gave earlier (at the end of Section 3) are indeed nd-sober.

We say that an nd-frame  $\mathcal{L}$  is *spatial* if there exists a bitopological space  $X$  such that  $\mathcal{L} \cong \Omega(X)$ . Again, similarly to d-frame theory, we have the (*co-unit*) map  $\epsilon_{\mathcal{L}}: \mathcal{L} \rightarrow \Omega\Sigma(\mathcal{L})$  defined as  $(a, b) \mapsto (\Phi_+(a), \Phi_-(b))$ . This, again, is natural in  $\mathcal{L}$ .

**Theorem 5.4** *For an nd-frame  $\mathcal{L}$ , the following are equivalent:*

- (i)  $\mathcal{L}$  is spatial.
- (ii)  $\mathcal{L} \cong \Omega\Sigma(\mathcal{L})$ .
- (iii) The co-unit  $\epsilon_{\mathcal{L}}$  is an isomorphism.
- (iv) The co-unit  $\epsilon_{\mathcal{L}}$  is injective, reflects **con** and **tot**, and  $\epsilon_{\mathcal{L}}^{-1}(\neg\alpha) \sqsubseteq \neg\epsilon_{\mathcal{L}}^{-1}(\alpha)$  for all  $\alpha \in \Omega\Sigma(\mathcal{L})$ .
- (v)  $\mathcal{L}$  satisfies the following conditions:
  - $(s_+)$   $\forall x \not\leq x' \in L_+ \exists (F_+, F_-) \in \Sigma(\mathcal{L}). x \in F_+, x' \notin F_+;$
  - $(s_-)$   $\forall y \not\leq y' \in L_- \exists (F_+, F_-) \in \Sigma(\mathcal{L}). y \in F_-, y' \notin F_-;$
  - $(s_{\text{con}})$   $\forall \alpha \notin \text{con} \exists (F_+, F_-) \in \Sigma(\mathcal{L}). \alpha_+ \in F_+, \alpha_- \in F_-;$
  - $(s_{\text{tot}})$   $\forall \alpha \notin \text{tot} \exists (F_+, F_-) \in \Sigma(\mathcal{L}). \alpha_+ \notin F_+, \alpha_- \notin F_-;$
  - $(s_p)$   $\forall a \not\leq p(x) \in L_+ \exists (F_+, F_-) \in \Sigma(\mathcal{L}). a \in F_+, x \notin F_-;$
  - $(s_m)$   $\forall b \not\leq m(y) \in L_- \exists (F_+, F_-) \in \Sigma(\mathcal{L}). y \notin F_+, b \in F_-.$

**Corollary 5.5**  *$\Omega$  and  $\Sigma$  form a (dual) adjunction with  $\eta$  and  $\epsilon$  being the unit and co-unit, respectively. Moreover, the restriction of  $\Sigma$  and  $\Omega$  to spatial nd-frames and nd-sober bispaces, respectively, forms a duality of categories.*

It is a good sign that extending Stone duality for d-frames to nd-frames is quite straightforward as it points towards the robustness of the theory. All the additional assumptions make sense topologically given that  $p$  and  $m$  should correspond to interior operators. The only difference with d-frame duality is that we added the conditions  $(dp_p)$  and  $(dp_m)$ . Similarly, in the characterisation of spatial nd-frames we needed to assume  $(s_p)$  and  $(s_m)$  in addition to the original conditions for spatial d-frames.

On the other hand, the language of nd-frames is definitely more expressive in

the sense that more bispaces are nd-sober than d-sober. We gave an example of this at the end of Section 3.

## 6 Canonical $(p, m)$

Every d-frame can be turned into an nd-frame in a trivial way, just augment  $(L_+ \times L_-; \text{con}, \text{tot})$  with  $p^{\text{triv}}$  and  $m^{\text{triv}}$  where  $p^{\text{triv}}$  and  $m^{\text{triv}}$  are trivial in the sense that they send 1 to 1 and everything else to 0. It is easy to see that this construction provides a left adjoint to the forgetful functor from **nd-Frm** to **d-Frm** that erases  $p$  and  $m$ .

The purpose of this section is to demonstrate that under mild conditions on a d-frame, a more interesting choice for  $p$  and  $m$  is available, one which interacts well with Stone duality. For motivation we begin by reviewing the notion of *regularity* for d-frames.

For a d-frame  $\mathcal{L} = (L_+ \times L_-; \text{con}, \text{tot})$  and for  $c, a \in L_+$  we say that  $c$  is *well-inside*  $a$  (and write  $c \triangleleft_+ a$ ) if there exists a  $d \in L_-$  such that  $(c, d) \in \text{con}$  and  $(a, d) \in \text{tot}$ . We define  $c \triangleleft_- a$  for  $c, a \in L_-$  dually, that is, with the roles of  $L_+$  and  $L_-$  switched. We say that  $\mathcal{L}$  is *d-regular* if

$$a = \bigvee \{c \in L_+ \mid c \triangleleft_+ a\} \quad \text{and} \quad b = \bigvee \{c \in L_- \mid c \triangleleft_- b\}$$

for all  $a \in L_+$  and  $b \in L_-$ . Finally, we say that a bitopological space  $X$  is *d-regular* if  $\Omega_d(X)$  is. Note that the well-inside relation has a clear bitopological reading. For  $U, V \in \tau_+$ ,  $U \triangleleft_+ V$  just means that  $\tau_-$ -closure of  $U$  is a subset of  $V$ .

We can express the interior operations of d-regular bispaces explicitly in the language of d-frames. Indeed, let  $X = (X; \tau_+, \tau_-)$  be d-regular. Then, for a  $U \in \tau_+$ ,

$$U^{\circ\tau_-} = \bigcup \{V \in \tau_- \mid \exists V' \in \tau_+. V \cap V' = \emptyset \text{ and } V' \cup U = X\}$$

Let  $\mathcal{L} = \Omega_d(X)$ , then the term above becomes, for an  $a \in L_+$ ,

$$m^r(a) = \bigvee \{x_- \in L_- \mid \exists x_+ \in L_+. (x_+, x_-) \in \text{con} \text{ and } x_+ \vee a = 1\}.$$

Notice the similarity of the relationship between  $x_-$  and  $a$  in this definition, and the well-inside relation defined above, except that here it is between elements from the two *different* components of  $\mathcal{L}$ . Also note that the definition of  $m^r(a)$  does not presuppose regularity of the underlying d-frame.

To simplify the definition of  $m^r$  a bit further, recall for any  $x \in L_-$  the *pseudo-complement*  $x^*$  of  $x$  is defined as  $\bigvee \{c \in L_+ \mid (c, x) \in \text{con}\}$ . This allows us to define our candidate interior operators as follows

$$m^r(a) = \bigvee \{x \in L_- \mid x^* \vee a = 1\} \quad \text{and} \quad p^r(b) = \bigvee \{x \in L_+ \mid x^* \vee b = 1\}$$

and to prove some of the required properties. To begin we see that  $m^r(1) = \bigvee \{x \mid x^* \vee 1 = 1\} \geq \bigvee \{1\} = 1$ . For the preservation of 0 recall that  $x^* = 1$  implies  $x = 0$  because  $(x^*, x) = (1, x) \in \text{con}$  and  $(1, 0) \in \text{tot}$  gives, by (con–tot),  $x \leq 0$ . Therefore  $m^r(0) = \bigvee \{x \mid x^* \vee 0 = 1\} = \bigvee \{0\} = 0$ .

It is clear that  $m^r$  is monotone, so in order to show that it preserves binary meets, it suffices to check that  $m^r(a \wedge a') \geq m^r(a) \wedge m^r(a')$  for all  $a, a' \in L_+$ . Assume  $x^* \vee a = 1$  and  $x'^* \vee a' = 1$ , then because pseudocomplement is antitone, we have for  $x'' = x \wedge x'$ ,  $x''^* \vee a = 1$  and  $x''^* \vee a' = 1$  from which it follows that  $x''^* \vee (a \wedge a') = 1$  and hence  $x'' \leq m^r(a \wedge a')$ . Frame distributivity now allows us to conclude the desired inequality.

We are also able to show  $(\text{con}-m)$ . For this let  $(a \wedge b, c) \in \text{con}$  and let  $x \in L_-$  be such that  $x^* \vee b = 1$ . Since,  $(x^*, x) \in \text{con}$ , by  $(\text{con}-\vee)$ , we have that  $(a \wedge b, c) \vee (x^*, x) \in \text{con}$ . Therefore, since  $\text{con}$  is  $\sqsubseteq$ -downwards closed,

$$(a \wedge b, c) \vee (x^*, x) = ((a \vee x^*) \wedge (b \vee x^*), c \wedge x) \supseteq (a, c \wedge x) \in \text{con}$$

where the inequality follows from  $x^* \vee b = 1$ . Since  $(a, c \wedge x) \in \text{con}$  for all  $x$  such that  $x^* \vee b = 1$ , then by  $(\text{con}-\bigsqcup^\uparrow)$  and frame distributivity we get that also  $(a, c \wedge m^r(b)) = (a, c \wedge \bigvee \{x \mid x^* \vee b = 1\}) \in \text{con}$ .

However, we can show neither  $(\text{pm}-4)$  nor  $(\text{tot}-p)$  at this level of generality. To make progress, recall the following two *infinitary cut rules* for d-frames (already discussed in [10], but not part of the definition of d-frames):

$$\frac{(x, y \vee \bigvee_{i \in I} b_i) \in \text{tot}, \forall i \in I. (x \vee a_i, y) \in \text{tot}, (a_i, b_i) \in \text{con}}{(x, y) \in \text{tot}} (\text{CUT}_r)$$

$$\frac{(x \vee \bigvee_{i \in I} a_i, y) \in \text{tot}, \forall i \in I. (x, y \vee b_i) \in \text{tot}, (a_i, b_i) \in \text{con}}{(x, y) \in \text{tot}} (\text{CUT}_l)$$

These two rules are precisely what we need to complete our construction:

**Proposition 6.1** *Let  $\mathcal{L} = (L_+ \times L_-; \text{con}, \text{tot})$  be a d-frame satisfying the infinitary cut rules. Then,  $(\mathcal{L}; p^r, m^r) = (L_+ \times L_-; \text{con}, \text{tot}; p^r, m^r)$  is an nd-frame.*

**Proof.** Only  $(\text{pm}-4)$  and  $(\text{tot}-p)$  remain to be shown. The former says that  $m^r p^r(b) \leq b$  for all  $b \in L_-$ . Since

$$m^r p^r(b) = \bigvee \{y \in L_- \mid y^* \vee p^r(b) = 1\},$$

it is enough to show that every  $y \in L_-$ , such that  $y^* \vee p^r(b) = 1$ , is less or equal to  $b$ . From the definition of  $p^r$  we have  $(1, 0) = (y^* \vee \bigvee \{x \mid x^* \vee b = 1\}, 0) \in \text{tot}$ , therefore, from  $(\text{tot}-\uparrow)$ , we get

- (1a)  $(y^* \vee \bigvee \{x \mid x^* \vee b = 1\}, b) \in \text{tot}$
- (2a) for all  $x$  such that  $x^* \vee b = 1$ :  $(y^*, x^* \vee b) \in \text{tot}$
- (3a)  $(x, x^*) \in \text{con}$ .

By applying  $(\text{CUT}_l)$  to (1a), (2a) and (3a) we obtain  $(y^*, b) \in \text{tot}$ , and from  $(\text{con}-\text{tot})$  that  $y \leq b$  as we wanted.

To show  $(\text{tot}-p)$ , let  $(a \vee p^r(b), c) \in \text{tot}$ . Again, by unwrapping the definitions we get  $(a \vee \bigvee \{x \mid x^* \vee b = 1\}, c) \in \text{tot}$  and this, by  $(\text{tot}-\uparrow)$ , gives us

- (1b)  $(a \vee \bigvee \{x \mid x^* \vee b = 1\}, b \vee c) \in \text{tot}$
- (2b) for all  $x$  such that  $x^* \vee b = 1$ :  $(a, x^* \vee b \vee c) \in \text{tot}$ .

$$(3b) \quad (x, x^*) \in \mathbf{con}.$$

Therefore,  $(\text{CUT}_I)$  applied to (1b), (2b) and (3b) gives us that  $(a, b \vee c) \in \mathbf{tot}$ .  $\square$

**Proposition 6.2** *The mapping  $\mathbf{N}: \mathbf{d-Frm}_{\text{CUT}} \rightarrow \mathbf{nd-Frm}$  assigning  $\mathcal{L} \mapsto (\mathcal{L}; p^r, m^r)$  is functorial, where  $\mathbf{d-Frm}_{\text{CUT}}$  is the category of d-frames satisfying the infinitary cut rules.*

**Proof.**  $\mathbf{N}$  is well defined on objects by Proposition 6.1. For morphisms, let  $h: \mathcal{L} \rightarrow \mathcal{M}$  be a d-frame homomorphism between two d-frames that satisfy the infinitary cut rules. We need to show that  $h(\neg\alpha) \sqsubseteq \neg h(\alpha)$  for all  $\alpha \in L_+ \times L_-$ . From the definition we see that the corresponding plus coordinates are computed as follows:

$$(h(\neg\alpha))_+ = h_+(p^r(\alpha_-)) = h_+(\bigvee\{x \mid x^* \vee \alpha_- = 1\}) = \bigvee\{h_+(x) \mid x^* \vee \alpha_- = 1\}$$

and

$$(\neg h(\alpha))_+ = p^r(h_-(\alpha_-)) = \bigvee\{w \mid w^* \vee h_-(\alpha_-) = 1\}.$$

It is sufficient to show that  $x^* \vee \alpha_- = 1$  implies  $h_+(x)^* \vee h_-(\alpha_-) = 1$ . This is true because from  $(x, x^*) \in \mathbf{con}$  we get that  $(h_+(x), h_-(x^*)) \in \mathbf{con}$  and hence  $h_+(x)^* \geq h_-(x^*)$ . Therefore, by applying the frame homomorphism  $h_-$  to  $x^* \vee \alpha_- = 1$  we obtain  $h_-(x^*) \vee h_-(\alpha_-) = 1$  and this implies  $h_+(x)^* \vee h_-(\alpha_-) = 1$ .  $\square$

**Remark 6.3** The d-frame of truth values  $\mathcal{FOUR}$  satisfies the cut rules (as it is spatial, for example) so we can also apply the functor  $\mathbf{N}$  to equip it with interior operators. However, only the identity maps  $\mathbf{2} \rightarrow \mathbf{2}$  are available, so this is what  $\mathbf{N}$  will produce.

### 6.1 Spectra and comparison with the interior operations

We are now ready to show that the spectrum of a  $(p^r, m^r)$  enriched d-frame is the same as the spectrum of the original d-frame.

**Proposition 6.4** *Let  $\mathcal{L}$  be a d-frame satisfying the infinitary cut rules. Then, the spectra of  $\mathcal{L}$  and  $(\mathcal{L}; p^r, m^r)$  are the same; that is*

$$\Sigma_d(\mathcal{L}) = \Sigma(\mathcal{L}; p^r, m^r).$$

**Proof.** This follows from the functoriality of  $\mathbf{N}$  as proved in Proposition 6.2: Every d-point of  $\mathcal{L}$  viewed as a d-frame homomorphism  $p: \mathcal{L} \rightarrow \mathcal{FOUR}$  is also an nd-point  $\mathbf{N}(p): (\mathcal{L}; p^r, m^r) \rightarrow \mathcal{FOUR}$ . The converse inclusion is immediate.  $\square$

From the fact that spatial d-frames satisfy the infinitary cut rules (Lemma 5.10 and Corollary 5.13 in [10]), we have:

**Corollary 6.5** *Let  $\mathcal{L}$  be a spatial d-frame. Then  $(\mathcal{L}; p^r, m^r)$  is an nd-frame and, moreover, the spectra of  $\mathcal{L}$  and  $(\mathcal{L}; p^r, m^r)$  are the same.*

Note that this still does not mean that, for a spatial d-frame  $\mathcal{L}$ ,  $p^r$  and  $m^r$  are the interior operations of the corresponding bispaces, but in the d-regular case everything works out:



**Proposition 6.6** *Let  $\mathcal{L}$  be a spatial d-regular d-frame. Then the nd-frame  $(\mathcal{L}; p^r, m^r)$  is spatial.*

**Proof.** We need to prove that the conditions  $(s_p)$  and  $(s_m)$  hold in  $(\mathcal{L}; p^r, m^r)$ . For  $(s_p)$ , assume  $a \not\leq p^r(b)$  for some  $a \in L_+$  and  $b \in L_-$ . From d-regularity, there exists  $c \in L_+$  such that  $(a, c^*) \in \text{tot}$  and  $c \not\leq p^r(b)$ . Since  $p^r(b) = \bigvee \{x \mid x^* \vee b = 1\}$ , the latter condition on  $c$  implies that  $c^* \vee b \neq 1$  or, in other words,  $(0, c^* \vee b) \notin \text{tot}$ . From spatiality of  $\mathcal{L}$ , there exists a point  $(F_+, F_-)$  such that  $c^* \vee b \notin F_-$  and, therefore, also  $c^* \notin F_-$  and  $b \notin F_-$ . Finally, we know that  $(a, c^*) \in \text{tot}$ , therefore it has to be the case that  $a \in F_+$ .  $\square$

### 6.2 Maximality of $(p^r, m^r)$

We saw before that d-regularity is enough to ensure that  $p^r$  and  $m^r$  correspond to the interior operations on the corresponding bispace. Here we show (assuming just d-regularity) that  $(p^r, m^r)$  is “larger” than any other  $(p, m)$  pair:

**Proposition 6.7** *Let  $\mathcal{L}$  be a d-regular d-frame and let  $(p, m)$  be such that  $(\mathcal{L}; p, m)$  is an nd-frame. Then,  $p \leq p^r$  and  $m \leq m^r$  in the pointwise order.*

**Proof.** Let  $b \in L_-$ . Since  $\mathcal{L}$  is d-regular,  $p(b) = \bigvee \{c \mid (p(b), c^*) \in \text{tot}\}$ . But, any time  $(p(b), c^*) \in \text{tot}$ , from  $(\text{tot}-p)$ , we also have that  $(0, c^* \vee b) \in \text{tot}$  and this is equivalent to  $c^* \vee b = 1$ . Therefore,  $p \leq p^r$ .  $\square$

The fact that  $(p^r, m^r)$  is maximal says that it frame-theoretically mimics the interior operations as closely as possible. Indeed, if  $(p^\circ, m^\circ)$  were the interior operators of a (spatial) d-regular d-frame then, since  $(p^\circ, m^\circ)$  satisfies the  $(p, m)$  axioms, the previous proposition says that  $(p^\circ, m^\circ)$  is pointwise smaller than  $(p^r, m^r)$ . On the other hand,  $p^r(b)$  is computed as a join of  $\tau_+$ -open elements well-inside  $b$ , whereas  $p^\circ(b)$  is computed as the join of *all*  $\tau_+$ -open subsets of  $b$ . Therefore,  $(p^r, m^r)$  is also pointwise smaller than  $(p^\circ, m^\circ)$ . This means that in the spatial case  $(p^r, m^r)$  and  $(p^\circ, m^\circ)$  coincide.

### 6.3 Proof-theoretic negation

Assuming the Gentzen cut rule in the original paper [10] allowed Jung and Moshier to give a proof-theoretic characterisation of negation. For a  $\gamma \in \mathcal{L}$ , let  $\bar{I}\gamma = \{\varphi \in \text{con} \mid \varphi \wedge \gamma \prec \text{ff}\}$  and  $\bar{F}\gamma = \{\psi \in \text{con} \mid \text{tt} \prec \gamma \vee \psi\}$ . Then, define the *proof-theoretic* negation of  $\gamma$  as

$$\bar{\gamma} \stackrel{\text{def}}{=} \left( \bigvee_{\varphi \in \bar{I}\gamma}^{\uparrow} \varphi_+, \bigvee_{\psi \in \bar{F}\gamma}^{\uparrow} \psi_- \right) \quad (\dagger)$$

We can now observe that this negation is actually exactly the same as the one obtained from the canonical  $(p^r, m^r)$ :

**Theorem 6.8** *Let  $\mathcal{L}$  be a d-frame. Then,  $\bar{\gamma} = (p^r(\gamma_-), m^r(\gamma_+))$  for all  $\gamma \in L_+ \times L_-$ .*

**Proof.** For “ $\sqsubseteq$ ”, let  $\varphi \in \bar{I}\gamma$ . Notice that  $\varphi \wedge \gamma \prec \text{ff}$  is equivalent to  $(0, \varphi_- \vee \gamma_-) \in \text{tot}$  which is the same as  $\varphi_- \vee \gamma_- = 1$ . Since  $\varphi \in \text{con}$ ,  $(\varphi_+)^* \geq \varphi_-$  and so  $(\varphi_+)^* \vee \gamma_- = 1$ ,

therefore  $\varphi_+ \leq p^r(\gamma_-)$ . Dually also  $\psi \in \overline{F}\gamma$  implies  $\psi_- \leq m^r(\gamma_+)$ . For “ $\sqsupseteq$ ”, let  $\varphi_+ \in L_+$  be such that  $(\varphi_+)^* \vee \gamma_- = 1$ . Define

$$\chi \stackrel{\text{def}}{=} (\varphi_+, (\varphi_+)^*).$$

Obviously  $\chi \in \mathbf{con}$  and  $\chi \wedge \gamma \prec \mathbf{ff}$  (this is exactly the condition  $\chi_- \vee \gamma_- = 1$ ). Therefore,  $\chi \in \overline{I}\gamma$  and  $\varphi_+ = \chi_+ \leq \overline{\gamma}_+$ . Dually, for every  $\varphi_- \in L_-$  such that  $(\varphi_-)^* \vee \gamma_+ = 1$ ,  $\varphi_- \leq \overline{\gamma}_-$  holds.  $\square$

In fact, the proof that  $m^r p^r \leq \text{id}$  and  $p^r m^r \leq \text{id}$  in Proposition 6.1 is a direct translation of the proof that  $\overline{\gamma} \sqsubseteq \gamma$  in [10]. The only, but very important, difference is that  $\overline{\gamma}$  was originally defined only for the consistent predicates whereas  $(p^r, m^r)$  is defined for the whole d-frame. On top of that, the previous theorem provides the proof-theoretic negation with a bitopological interpretation.

## Acknowledgements

We would like to thank the referees of the MFPS 32 conference for their detailed comments on an earlier version of this paper.

## References

- [1] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.
- [2] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Semantic Structures*, volume 3 of *Handbook of Logic in Computer Science*, pages 1–168. Clarendon Press, 1994.
- [3] O. Arieli and A. Avron. Reasoning with logical bilattices. *Journal of Logic, Language and Information*, 5:25–63, 1996.
- [4] A. Avron. The structure of interlaced bilattices. *Mathematical Structures in Computer Science*, 6:287–299, 1996.
- [5] N. D. Belnap. A useful four-valued logic. In J. M. Dunn and G. Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 8–37. Reidel Publishing Company, 1977.
- [6] Félix Bou, Ramon Jansana, and Umberto Rivieccio. Varieties of interlaced bilattices. *Algebra universalis*, 66(1):115–141, 2011.
- [7] M. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.
- [8] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *Continuous Lattices and Domains*, volume 93 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2003.
- [9] M. Ginsberg. Multivalued logics: A uniform approach to inference in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.
- [10] A. Jung and M. A. Moshier. On the bitopological nature of Stone duality. Technical Report CSR-06-13, School of Computer Science, The University of Birmingham, 2006. 110 pages.
- [11] B. Mobasher, D. Pigozzi, G. Slutzki, and G. Voutsadakis. A duality theory for bilattices. *Algebra Universalis*, 43:109–125, 2000.
- [12] Jorge Picado and Aleš Pultr. *Frames and Locales: Topology without points*. Springer-Birkhäuser Basel, 2011.
- [13] G. Priest. Paraconsistent logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume 6, pages 287–393. Kluwer Academic Publishers, 2nd edition, 2002.
- [14] U. Rivieccio. *An Algebraic Study of Bilattice-based Logics*. PhD thesis, University of Barcelona, 2010.
- [15] M. B. Smyth. Powerdomains. *Journal of Computer and Systems Sciences*, 16:23–36, 1978.

## A Appendix: Proofs omitted from the main text

### A.1 Proofs related to Remark 4.3

**Proposition A.1** *Satisfying any one of the following axioms is equivalent to requiring that  $p \circ m = \text{id}$ :*

$$\begin{aligned} (\neg\neg L) \quad & \neg\neg\varphi \subset \varphi \\ (\neg\supset L) \quad & \neg(\varphi \supset \psi) \subset \varphi \wedge \neg\psi \end{aligned}$$

*and satisfying any one of the following axioms is equivalent to requiring that  $p$  preserves finite suprema:*

$$\begin{aligned} (\neg\wedge R) \quad & \neg(\varphi \wedge \psi) \supset \neg\varphi \vee \neg\psi \\ (\neg\sqcup R) \quad & \neg(\varphi \sqcup \psi) \supset \neg\varphi \sqcup \neg\psi \end{aligned}$$

**Proof.**  $(\neg\neg L)$  is equivalent to  $p \circ m = \text{id}$  as, by Lemma 4.1 (L2),  $\mathcal{L} \models \varphi \supset \neg\neg\varphi$  is equivalent to  $v(\varphi)_+ \leq v(\neg\neg\varphi)_+ = p(m(v(\varphi)_+))$  for all valuations  $v$ . To show the same for  $(\neg\supset L)$ , again by Lemma 4.1 (L2),  $\mathcal{L} \models \varphi \wedge \neg\psi \supset \neg(\varphi \supset \psi)$  is equivalent to  $v(\varphi)_+ \wedge p(v(\psi)_-) \leq p(m(v(\varphi)_+)) \wedge p(v(\psi)_-)$  for all valuations  $v$ . This has to hold for all  $\varphi$  and  $\psi$ . If  $\psi$  is such that  $\psi_- = 1$  we get  $v(\varphi)_+ \wedge p(1) \leq p(m(v(\varphi)_+)) \wedge p(1)$  and, since both  $p$  and  $m$  preserve 1, we get  $v(\varphi)_+ \leq p(m(v(\varphi)_+))$  as we wanted.

For the second part we use Lemma 4.1 (L2) once again to show that  $(\neg\wedge R)$  is equivalent to  $p(\varphi_- \vee \psi_-) = p(\varphi_-) \vee p(\psi_-)$  for all  $\varphi$  and  $\psi$ . Notice that,  $p(x \vee y) \geq p(x) \vee p(y)$  holds always from monotonicity of  $p$ , and  $\mathcal{L} \models \neg(\varphi \wedge \psi) \supset \neg\varphi \vee \neg\psi$  is equivalent to  $p(v(\varphi)_- \vee v(\psi)_-) \leq p(v(\varphi)_-) \vee p(v(\psi)_-)$  for all valuations  $v$ . The same argument applies for  $(\neg\sqcup R)$ .  $\square$

We can also assume the following axiom

$$(\supset 3) \quad (\varphi \supset \psi) \supset \varphi \supset \varphi$$

which is equivalent to assuming that  $L_+$  is a Boolean algebra as it forces  $((v(\varphi)_+ \rightarrow v(\psi)_+) \rightarrow v(\varphi)_+) \rightarrow v(\varphi)_+ = 1$  to hold for all valuations  $v$ .

### A.2 Proofs of the main theorems in Section 5

**Lemma A.2** *Let  $\mathcal{L}$  be an  $nd$ -frame. Then,  $(\Phi_+(x))^{\circ\tau-} \supseteq \Phi_-(m(x))$  and  $(\Phi_-(y))^{\circ\tau+} \supseteq \Phi_+(p(y))$  in  $\Sigma(\mathcal{L})$ , for all  $x \in L_+$  and  $y \in L_-$ .*

**Proof.** The second statement is true because, from  $(\text{dp}_p)$ ,  $\Phi_+(p(y)) \subseteq \Phi_-(y)$ .  $\square$

**Proof of Theorem 5.3 (following the proof of Theorem 4.1 in [10]).** Clearly, (iii) implies (ii) which implies (i). As in the original paper,  $\eta_X$  is bicontinuous, biopen onto the image and natural in  $X$  we see that (iii) and (iv) are equivalent. To show that (i) implies (iv) assume that  $\iota: X \cong \Sigma(\mathcal{L})$ . If we prove that  $\eta_{\Sigma(\mathcal{L})}$  is a bijection, we get that also  $\eta_X$  is because the following square commutes (from naturality of  $\eta$ ):

$$\begin{array}{ccc}
 X & \xrightarrow{\iota} & \Sigma(\mathcal{L}) \\
 \downarrow \eta_X & & \downarrow \eta_{\Sigma(\mathcal{L})} \\
 \Sigma\Omega(X) & \xrightarrow{\Sigma\Omega(\iota)} & \Sigma\Omega\Sigma(L)
 \end{array}$$

Observe that  $\eta_{\Sigma(\mathcal{L})}$  is injective for all  $\mathcal{L}$ . For the surjectivity of  $\eta_{\Sigma(\mathcal{L})}$ , take a  $(\mathcal{F}_+, \mathcal{F}_-) \in \Sigma\Omega\Sigma(L)$  and define

$$F_+ = \{x \in L_+ \mid \Phi_+(x) \in \mathcal{F}_+\} \quad F_- = \{y \in L_- \mid \Phi_-(y) \in \mathcal{F}_-\}.$$

We will show that  $(F_+, F_-)$  is an nd-point of  $L$ . As in [10], the pair  $(F_+, F_-)$  is a d-point. To show that it is actually an nd-point, we need to show that it satisfies  $(\text{dp}_p)$  and  $(\text{dp}_m)$ . For the former, let  $p(x) \in F_+$ . This is equivalent to  $\Phi_+(p(x)) \in \mathcal{F}_+$ . From Lemma A.2 we also know that  $\Phi_+(p(x)) \subseteq (\Phi_-(x))^{\circ\tau_+} \in \mathcal{F}_+$ . And, since  $(\mathcal{F}_+, \mathcal{F}_-)$  is an nd-point, we know that  $\Phi_-(x) \in \mathcal{F}_-$ . Finally, from the definition of  $F_-$  we get that  $x \in F_-$  as we wanted. The proof of  $(\text{dp}_m)$  is the same but dual.

The argument that  $\eta_{\Sigma(\mathcal{L})}(F_+, F_-) = (\mathcal{F}_+, \mathcal{F}_-)$  is exactly the same as in [10].  $\square$

**Lemma A.3**  $\epsilon_{\mathcal{L}}$  is an onto nd-frame homomorphism. Moreover,  $\epsilon$  is natural in  $\mathcal{L}$ .

**Proof.** Since  $\epsilon$  is defined the same way as for d-frames, we see that  $\epsilon_{\mathcal{L}}$  is a d-frame homomorphism and that  $\epsilon$  is natural in  $\mathcal{L}$ . We need to show that it is indeed an nd-frame homomorphism. That is that it satisfies  $\epsilon_{\mathcal{L}}(\neg\alpha) \sqsubseteq \neg\epsilon_{\mathcal{L}}(\alpha)$  for all  $\alpha \in L_+ \times L_-$  but this is exactly the same statement as Lemma A.2.  $\square$

**Proof of Theorem 5.4 (following the proof of Theorem 5.1 in [10]).** The implications (iii)  $\implies$  (ii)  $\implies$  (i) are immediate, and (iv)  $\implies$  (iii) follows from the fact that  $\epsilon_L$  is onto (by Lemma A.3).

To show that (i) implies (v), it is enough to show that  $(s_p)$  and  $(s_m)$  hold for all images of  $\Omega$  (the other conditions were already proved in [10] for d-frames). Clearly, for  $U_+ \in \tau_+$  and  $V_- \in \tau_-$ ,  $U_+ \subseteq V_-^{\circ\tau_+}$  iff  $U_+ \subseteq V_-$ . Therefore, if  $U_+ \not\subseteq V_-^{\circ\tau_+}$ , then there exists an  $x \in U_+ \setminus V_-$  such that  $U_+ \in \mathcal{U}_+(x)$  and  $V_- \notin \mathcal{U}_-(x)$ . Moreover,  $(\mathcal{U}_+(x), \mathcal{U}_-(x))$  is an nd-point.

Finally, (v) implies (iv). As already proved in [10],  $\epsilon_L$  is injective and reflects con and tot. It remains to prove that  $\epsilon_L^{-1}(\neg(\Phi_+(x), \Phi_-(y))) \sqsubseteq \neg\epsilon_L^{-1}(\Phi_+(x), \Phi_-(y))$ . Let us focus on the plus coordinates, that is to prove that

$$(\epsilon_L)_+^{-1}(\Phi_-(y)^{\circ\tau_+}) \leq p(\epsilon_L)_-^{-1}(\Phi_-(y)).$$

Observe that  $\Phi_-(y)^{\circ\tau_+} = \Phi_+(x)$  for some  $x \in L_+$  because  $\epsilon_{\mathcal{L}}$  is onto. Since  $\epsilon_L$  is injective,  $(\epsilon_L)_+^{-1}(\Phi_-(y)^{\circ\tau_+}) = (\epsilon_L)_+^{-1}(\Phi_+(x)) = x$  and also  $p((\epsilon_L)_-^{-1}(\Phi_-(y))) = p(y)$ . Now, assume for a contradiction that  $x \not\leq p(y)$ . Then, from  $(s_p)$ , there exists an nd-point  $(F_+, F_-)$  such that  $x \in F_+$  and  $y \notin F_-$  but this is impossible since  $\Phi_+(x) = \Phi_-(y)^{\circ\tau_+} \subseteq \Phi_-(y)$ .  $\square$

# Effectuses from Monads

Bart Jacobs

*Institute for Computing and Information Sciences (iCIS),  
Radboud University Nijmegen, The Netherlands.  
Web address: [www.cs.ru.nl/B.Jacobs](http://www.cs.ru.nl/B.Jacobs)*

*May 20, 2016*

---

## Abstract

Effectuses have recently been introduced as categorical models for quantum computation, with probabilistic and Boolean (classical) computation as special cases. These ‘probabilistic’ models are called commutative effectuses. All known examples of such commutative effectuses are Kleisli categories of a monad. This paper answers the open question what properties a monad should satisfy so that its Kleisli category is a (commutative) effectus. The relevant properties are: strong affineness and partial additivity, together with some non-triviality conditions.

*Keywords:* monad, effectus, probabilistic computation

---

## 1 Introduction

An effectus is a relatively simple category, with finite coproducts and a final object, satisfying some elementary properties: certain squares have to be pullbacks and certain parallel maps have to be jointly monic, see (9) and (8) below. These effectuses have been introduced in [8], and give rise to a rich theory that includes quantum computation, see the overview paper [4]. Subclasses of ‘commutative’ effectuses and ‘Boolean’ effectuses have been identified. These Boolean effectuses capture classical (deterministic) computation, and can be characterised as extensive categories, see [4, Sec. 13] for details. This is a non-trivial result. A similar result for commutative effectuses is still missing. It should lead to a characterisation of (categorical) models of probabilistic computation.

This paper builds on [9] and makes a significant step towards a conjectured characterisation of these commutative effectuses as Kleisli categories of certain monads. The main result of this paper says that if the monad is strongly affine and partially additive, then its Kleisli category is an effectus. Affineness of a monad  $T$  means that it preserves the final object:  $T(1) \cong 1$ . The property ‘strong affineness’ comes from [9], where it is used to prove that it yields a bijective correspondence between

---

<sup>1</sup> [bart@cs.ru.nl](mailto:bart@cs.ru.nl)

predicates and side-effect-free instruments (as in a non-quantum settings). Partial additivity of a monad has been introduced in [7] where it is used to obtain partially additive structure on homsets of a Kleisli category. This result is re-used here, as a step towards constructing effectuses, following [3].

We describe five monads to which our main result applies: distribution, Giry, probabilistic powerdomain, Radon, and expectation. These monads are all ‘probabilistic’ in an intuitive sense, and their Kleisli categories are (commutative) effectuses. In future work we hope to find a construction in the other direction, turning a commutative effectus, possibly satisfying some additional properties, into a ‘probabilistic’ monad.

This paper is organised as follows. After some preliminary remarks about categories and monads in Section 2 – 4 we describe the properties of strong affineness and partial additivity of monads in Section 5. Our main result, Theorem 6.3, stating conditions on a monad that make its Kleisli category an effectus, is in Section 6. Subsequently, Section 7 shows in some details that the requirements hold for two of the monad examples, namely the probabilistic power monad and the Radon monad.

## 2 Categorical preliminaries

This section briefly describes our assumptions about the underlying category that we will be using. It is a distributive category, which is non-trivial in a suitable sense that will be explained below. We recall from [5] that coprojections  $\kappa_i: X_i \rightarrow X_1 + X_2$  in a distributive category are monic, and that the initial object 0 is strict — that is, each map  $X \rightarrow 0$  is an isomorphism.

**Definition 2.1** A category is called *distributive* if it has finite products  $(\times, 1)$  and coproducts  $(+, 0)$ , where products distribute over coproducts, in the sense that the following maps are isomorphisms.

$$0 \xrightarrow{!} 0 \times X \quad (A \times X) + (B \times X) \xrightarrow{\text{dis}_1 = [\kappa_1 \times \text{id}, \kappa_2 \times \text{id}]} (A + B) \times X \quad (1)$$

We call such a distributive category *non-trivial* if it satisfies the following two additional requirements.

- (i) For each object  $X$  we have:  $X \not\cong 0$  iff there is a map  $x: 1 \rightarrow X$ . This implies  $1 \not\cong 0$ .
- (ii) The coprojections  $\kappa_1, \kappa_2: 1 \rightarrow 1 + 1$  are disjoint, *i.e.* form a pullback:

$$\begin{array}{ccc} 0 & \longrightarrow & 1 \\ \downarrow & \lrcorner & \downarrow \kappa_2 \\ 1 & \xrightarrow{\kappa_1} & 1 + 1 \end{array} \quad (2)$$

This implies  $1 + 1 \not\cong 1$ , or equivalently,  $\kappa_1 \neq \kappa_2$ , using point (i).

Swapping the distributivity map  $\text{dis}_1$  in (1) yields an associated distributivity map:

$$(X \times A) + (X \times B) \xrightarrow[\text{= } \gamma \circ \text{dis}_1 \circ (\gamma + \gamma)]{\text{dis}_2 = [\text{id} \times \kappa_1, \text{id} \times \kappa_2]} X \times (A + B)$$

where  $\gamma = \langle \pi_2, \pi_1 \rangle$  is the (product) swap isomorphism.

### 3 Monad preliminaries

In this paper we will be working with a monad  $T = (T, \eta, \mu)$  on a non-trivial distributive category  $\mathbf{C}$ . This section describes the notation and terminology that we use for monads.

We shall write  $\mathcal{Kl}(T)$  for the Kleisli category of the monad  $T$ , and  $\bullet$  for Kleisli composition, that is, for composition in  $\mathcal{Kl}(T)$ , in order to distinguish it from composition  $\circ$  in the underlying category  $\mathbf{C}$ . Explicitly for ‘Kleisli’ maps  $f: X \rightarrow Y$  and  $g: Y \rightarrow Z$  in  $\mathcal{Kl}(T)$  we have  $g \bullet f = \mu \circ T(g) \circ f: X \rightarrow T(Y) \rightarrow T^2(Z) \rightarrow T(Z)$ . The identity map on an object  $X \in \mathcal{Kl}(T)$  is given by the unit map  $\eta: X \rightarrow T(X)$ . Each map  $f: X \rightarrow Y$  in  $\mathbf{C}$  yields a map  $\langle f \rangle = \eta \circ f: X \rightarrow T(X)$  in  $\mathcal{Kl}(T)$ . This gives a functor  $\langle - \rangle: \mathbf{C} \rightarrow \mathcal{Kl}(T)$ .

The Kleisli category  $\mathcal{Kl}(T)$  inherits coproducts  $(+, 0)$  from  $\mathbf{C}$ , with coprojections of the form  $\langle \kappa_i \rangle: X_i \rightarrow X_1 + X_2$ . We call the monad  $T$  *non-trivial* if, in analogy with diagram (2), the following rectangle is a pullback in  $\mathbf{C}$ .

$$\begin{array}{ccc} 0 & \xrightarrow{\quad} & T(1) \\ \downarrow \lrcorner & & \downarrow T(\kappa_2) \\ T(1) & \xrightarrow{T(\kappa_1)} & T(1+1) \end{array} \quad (3)$$

In the terminology that will be used later, this says that the scalars 1 and 0 are not the same.

The *lift* monad  $(-) + 1$  exists not only on  $\mathbf{C}$ , but also on  $\mathcal{Kl}(T)$ , with unit and multiplication of the latter described in  $\mathbf{C}$  as:

$$X \xrightarrow{\langle \kappa_1 \rangle} T(X+1) \quad (X+1) + 1 \xrightarrow{\langle [\text{id}, \langle \kappa_2 \rangle] \rangle} T(X+1)$$

These maps are obtained via the functor  $\langle - \rangle$  from the unit and multiplication of the lift monad  $(-) + 1$  on  $\mathbf{C}$ . It is not hard to see that the Kleisli category of the lift monad  $(-) + 1$  on  $\mathcal{Kl}(T)$  is the Kleisli category of the monad  $T' = T((-) + 1)$  on  $\mathbf{C}$ . Hence we consider the category  $\mathcal{Kl}(T')$  as the category of *partial maps* in  $\mathcal{Kl}(T)$ . The unit  $\eta'$  and multiplication  $\mu'$  of  $T'$  are given by:

$$X \xrightarrow[\langle \kappa_1 \rangle]{\eta' =} T(X+1) \quad T(T(X+1) + 1) \xrightarrow[\mu \circ T([\text{id}, \langle \kappa_2 \rangle])]{\mu' =} T(X+1)$$

Abstractly, this  $T'$  is monad since there is always a distributive law of monads  $T(-) + 1 \Rightarrow T(- + 1)$ . In general, given such a law  $ST \Rightarrow TS$ , the composite  $TS$  is a monad again. Moreover, the monad  $S$  can be lifted to a monad  $\bar{S}$  on  $\mathcal{Kl}(T)$ , and its Kleisli category  $\mathcal{Kl}(\bar{S})$  is the same as the Kleisli category  $\mathcal{Kl}(TS)$  of the composite monad.

Kleisli composition in  $\mathcal{Kl}(T')$ , written as  $\bullet'$ , is related to composition  $\bullet$  in  $\mathcal{Kl}(T)$  via:

$$\begin{aligned} g \bullet' f &= \mu' \circ T'(g) \circ f = \mu \circ T([\text{id}, \langle \kappa_2 \rangle]) \circ T(g + \text{id}) \circ f \\ &= \mu \circ T([g, \langle \kappa_2 \rangle]) \circ f = [g, \langle \kappa_2 \rangle] \bullet f. \end{aligned}$$

To summarise, we will be working with three different categories with identity and

composition notation as described below.

$$(\mathbf{C}, \text{id}, \circ) \quad (\mathcal{Kl}(T), \eta, \bullet) \quad (\mathcal{Kl}(T'), \langle \kappa_1 \rangle, \bullet').$$

## 4 Monad examples

There are (at least) five monad that are of interest in the current setting: the distribution monad  $\mathcal{D}$  on sets, the Giry monad  $\mathcal{G}$  on measurable spaces, the expectation monad  $\mathcal{E}$  on sets, the probabilistic powerdomain monad  $\mathcal{V}$  on (continuous) dcpos, and the Radon monad  $\mathcal{R}$  on compact Hausdorff spaces. Due to space restrictions we will only elaborate the last two examples and refer to [9] for the first three.

### 4.1 The probabilistic powerdomain monad $\mathcal{V}$ on $\mathbf{Dcpo}$

We write  $\mathbf{Dcpo}$  for the category of directed complete partial orders (dcpo's), with (Scott) continuous functions between them. For a dcpo  $X$  we write  $\mathcal{O}(X)$  for the complete lattice of Scott open subsets: upward closed subsets  $U \subseteq X$  with: if  $\bigvee_i x_i \in U$ , then  $x_i \in U$  for some index  $i$ . A *valuation* on the dcpo  $X$  is a Scott continuous map  $\phi: \mathcal{O}(X) \rightarrow [0, 1]$  which satisfies  $\phi(\emptyset) = 0$ ,  $\phi(X) = 1$ , and  $\phi(U \cup V) = \phi(U) + \phi(V) - \phi(U \cap V)$  for all opens  $U, V$ . The requirement  $\phi(X) = 1$  means that valuations as used here are normalised. Without this requirement we speak of ‘sub-valuations’; they are standardly used in the theory of probabilistic powerdomains. We prefer to use proper, normalised valuations to obtain affineness, see below.

We write  $\mathcal{V}(X)$  for the set of valuations on a dcpo  $X$ , ordered pointwise, with pointwise directed joins. This yields a dcpo again, and an endofunctor  $\mathcal{V}: \mathbf{Dcpo} \rightarrow \mathbf{Dcpo}$ , where  $\mathcal{V}(f)(\phi)(V) = \phi(f^{-1}(V))$ , for  $f: X \rightarrow Y$ ,  $\phi \in \mathcal{V}(X)$  and  $V \in \mathcal{O}(Y)$ . This functor restricts to the category  $\mathbf{Cdcpo}$  of *continuous* dcpo's, see [11, Thm. 8.2], where each element is a directed joint of elements way below it.

It is not hard to see that  $\mathcal{V}(1) \cong 1$  and  $\mathcal{V}(2) \cong [0, 1]$ . A predicate on  $X$  is a map  $X \rightarrow 2$  in the Kleisli category  $\mathcal{Kl}(\mathcal{V})$ , and thus corresponds to a continuous function  $p: X \rightarrow [0, 1]$ . Given a valuation  $\phi: \mathcal{O}(X) \rightarrow [0, 1]$  on  $X$  one can define an integral  $\int p \, d\phi \in [0, 1]$  as join of integrals of simple functions, see [10, 11] for details.

This  $\mathcal{V}$  forms a monad [10, 11] on (continuous) dcpo's, that is, on both the categories  $\mathbf{Dcpo}$  and  $\mathbf{Cdcpo}$ . The unit  $\eta: X \rightarrow \mathcal{V}(X)$  is given by  $\eta(x)(U) = \mathbf{1}_U(x)$ , where  $\mathbf{1}_U: X \rightarrow [0, 1]$  is the indicator function for  $U$ , with  $\mathbf{1}_U(x) = 1$  if  $x \in U$  and  $\mathbf{1}_U(x) = 0$  otherwise. The Kleisli extension  $f_*: \mathcal{V}(X) \rightarrow \mathcal{V}(Y)$  of a continuous map  $f: X \rightarrow Y$  is given by integration:  $f_*(\phi)(V) = \int f(-)(V) \, d\phi$ .

This monad  $\mathcal{V}$  is strong, with strength map  $\text{st}_1: \mathcal{V}(X) \times Y \rightarrow \mathcal{V}(X \times Y)$  given by  $\text{st}_1(\phi, y)(U \times V) = \phi(U) \cdot \mathbf{1}_V(y)$ . The induced ‘double’ strength  $\text{dst}: \mathcal{V}(X) \times \mathcal{V}(Y) \rightarrow \mathcal{V}(X \times Y)$  is given by  $\text{dst}(\phi, \psi)(U \times V) = \phi(U) \cdot \psi(V)$ . This  $\mathcal{V}$  is a commutative monad, by Fubini for  $\mathcal{V}$ .

### 4.2 The Radon monad $\mathcal{R}$ on $\mathbf{CH}$

We can only describe the essentials of the Radon monad  $\mathcal{R}$  on the category  $\mathbf{CH}$  of compact Hausdorff spaces (with continuous maps) and refer to [6] for more in-



formation. This monad sends a compact Hausdorff space  $X$  to the states on the associated commutative  $C^*$ -algebra  $C(X)$  of continuous functions  $X \rightarrow \mathbb{C}$ . Hence we write  $\mathcal{R}(X) = \text{Stat}(C(X))$ , where states are positive unital maps  $\omega: C(X) \rightarrow \mathbb{C}$ . This  $\mathcal{R}$  is a ‘double dual’ monad, with familiar unit and Kleisli extension definitions:

$$\eta(x)(\phi) = \phi(x) \quad \text{and} \quad f_*(\omega)(\psi) = \omega(\lambda x. f(x)(\psi)) \quad \text{for } f: X \rightarrow \mathcal{R}(Y).$$

One of the main results of [6] — presented as a probabilistic version of Gelfand duality — states that the Kleisli category  $\mathcal{Kl}(\mathcal{R})$  is the opposite  $(\mathbf{CCstar}_{\text{PU}})^{\text{op}}$  of the the category of commutative  $C^*$ -algebras, with positive unital maps between them. This is a prime example of a (commutative) effectus, see [8,4].

## 5 Monad requirements

In this section we assume that  $T$  is a monad on a distributive category  $\mathbf{C}$ . As before, we write  $T' = T((-) + 1)$  for the induced monad on  $\mathbf{C}$ .

**Definition 5.1** The monad  $T$  is called *affine* if  $T(1) \cong 1$ , and *strongly affine* if  $T$  is strong and all rectangles below are pullbacks.

$$\begin{array}{ccc} T(X) \times Y & \xrightarrow{\pi_2} & Y \\ \text{st}_1 \downarrow \lrcorner & & \downarrow \eta_Y \\ T(X \times Y) & \xrightarrow{T(\pi_2)} & T(Y) \end{array} \quad (4)$$

It is not hard to see that a strongly affine monad is affine, see [9] for details.

The following result forms the technical core of this paper.

**Lemma 5.2** *Let  $T$  be strongly affine monad on a non-trivial distributive category  $\mathbf{C}$ . The following diagrams are then pullbacks in the Kleisli category  $\mathcal{Kl}(T)$ .*

$$\begin{array}{ccc} X \xrightarrow{!} 1 & & 1 \xlongequal{\quad} 1 \\ \langle \kappa_i \rangle \downarrow & & \downarrow \langle \kappa_2 \rangle \\ X + X \xrightarrow{!+!} 1 + 1 & & X + 1 \xrightarrow{!+\text{id}} 1 + 1 \end{array} \quad \begin{array}{ccc} X \xrightarrow{!} 1 & & 1 \xlongequal{\quad} 1 \\ \langle \kappa_1 \rangle \downarrow & & \downarrow \langle \kappa_2 \rangle \\ X + 1 \xrightarrow{!+\text{id}} 1 + 1 & & X + 1 \xrightarrow{!+\text{id}} 1 + 1 \end{array} \quad (5)$$

For this last (third) pullback we need to assume that the monad  $T$  is non-trivial. We can then prove that maps  $T(\kappa_i)$  are monic in  $\mathbf{C}$  — making coprojections  $\langle \kappa_i \rangle$  monic in  $\mathcal{Kl}(T)$ .

**Proof** The proof that the diagram on the left in (5) is a pullback is obtained by taking  $Y = 2 = 1 + 1$  in Diagram (4) and using the distributivity isomorphism  $\text{sep}_2 = (\pi_1 + \pi_1) \circ \text{dis}_2^{-1}: X \times 2 \rightarrow X \times 1 + X \times 1 \rightarrow X + X$ . We leave it to the meticulous reader to check that the following two diagrams commute.

$$\begin{array}{ccc} X \times 1 \xrightarrow{\text{id} \times \kappa_1} X \times 2 & & T(X) \times 2 \xrightarrow{\text{st}_1} T(X \times 2) \\ \pi_1 \downarrow & \searrow \text{sep}_2 \quad \pi_2 \searrow & \downarrow \text{sep}_2 \quad \downarrow T(\text{sep}_2) \\ X \xrightarrow{\kappa_1} X + X \xrightarrow{!+!} 2 & & T(X) + T(X) \xrightarrow{[T(\kappa_1), T(\kappa_2)]} T(X + X) \end{array} \quad (*)$$

We now show that the left diagram in (5) is a pullback in  $\mathcal{Kl}(T)$ , for  $i = 1$ . Let  $f: Y \rightarrow T(X + X)$  satisfy  $(! + !) \bullet f = \langle \kappa_1 \rangle \bullet !$ , that is,  $T(! + !) \circ f = T(\kappa_1) \circ \eta \circ !$ . Take  $f' = T(\text{sep}_2^{-1}) \circ f: Y \rightarrow T(X \times 2)$ , and consider the pullback (4). We get:

$$T(\pi_2) \circ f' = T(\pi_2) \circ T(\text{sep}_2^{-1}) \circ f \stackrel{(*)}{=} T(! + !) \circ f = \eta \circ \kappa_1 \circ !.$$

Hence there is a unique map  $g: Y \rightarrow T(X)$  in (4) with  $\text{st}_1 \circ \langle g, \kappa_1 \circ ! \rangle = f'$ . This  $g$  is the mediating map that we want, since:

$$\begin{aligned} f &= T(\text{sep}_2) \circ f' = T(\text{sep}_2) \circ \text{st}_1 \circ \langle g, \kappa_1 \circ ! \rangle \\ &\stackrel{(*)}{=} [T(\kappa_1), T(\kappa_2)] \circ \text{sep}_2 \circ (\text{id} \times \kappa_1) \circ \langle g, ! \rangle \\ &\stackrel{(*)}{=} [T(\kappa_1), T(\kappa_2)] \circ \kappa_1 \circ \pi_1 \circ \langle g, ! \rangle \\ &= T(\kappa_1) \circ g \\ &= \langle \kappa_1 \rangle \bullet g. \end{aligned}$$

Uniqueness is left to the reader.

We continue with the diagram in the middle in (5). The case  $X \cong 0$  trivially holds. If  $X \not\cong 0$ , then we may assume a map  $x: 1 \rightarrow X$ , since the underlying category is non-trivial, see Definition 2.1 (i). Now let  $f: Y \rightarrow T(X + 1)$  satisfy  $T(! + \text{id}) \circ f = \langle \kappa_2 \rangle \circ !$ . Then  $f' = T(\text{id} + x) \circ f: Y \rightarrow T(X + X)$  satisfies  $T(! + !) \circ f' = T(! + \text{id}) \circ f = \langle \kappa_2 \rangle \circ !$ . Using the pullback on the left in (5) we get a  $g: Y \rightarrow T(X)$  with  $T(\kappa_2) \circ g = f'$ . But then:

$$\begin{aligned} f &= T(\text{id} + !) \circ f' = T(\text{id} + !) \circ T(\kappa_2) \circ g = T(\kappa_2) \circ T(!) \circ g \\ &\stackrel{(**)}{=} T(\kappa_2) \circ \eta \circ ! = \langle \kappa_2 \rangle \bullet !. \end{aligned}$$

The equation  $\stackrel{(**)}{=}$  holds because  $T(1)$  is final. This finality also yields uniqueness of the mediating map  $!$ .

For the third rectangle in (5) the case  $X \cong 0$  is covered by the requirement that  $T$  is non-trivial: if  $f: Y \rightarrow T(0 + 1)$  satisfies  $T(! + \text{id}) \circ f = T(\kappa_1) \circ \eta \circ !$ , then  $f = T(\kappa_2) \circ \eta \circ !$ , since  $T(0 + 1) \cong T(1) \cong 1$ . We thus have  $T(\kappa_1) \circ \eta \circ ! = T(\kappa_2) \circ \eta \circ !$ , so that  $Y \rightarrow T(1)$  factors through 0, via the pullback (3). This implies  $Y \cong 0$ , since the initial object in a distributive category is strict [5]. But then we are done.

When  $X \not\cong 0$  we can use a map  $x: 1 \rightarrow X$  and proceed like for the middle rectangle. Finally, we show that the maps  $T(\kappa_1): T(X) \rightarrow T(X + Y)$  are monic in  $\mathbf{C}$ . If  $f, g: Y \rightarrow T(X)$  satisfy  $T(\kappa_1) \circ f = T(\kappa_1) \circ g$ , then  $f = g$  by uniqueness of the mediating map in the pullback on the right in (5). Obviously,  $! \bullet f = ! \bullet g$ , but also:

$$\begin{aligned} \langle \kappa_1 \rangle \bullet f &= T(\kappa_1) \circ f = T(\text{id} + !) \circ T(\kappa_1) \circ f \\ &= T(\text{id} + !) \circ T(\kappa_1) \circ g = T(\kappa_1) \circ f = \langle \kappa_1 \rangle \bullet g. \quad \square \end{aligned}$$

If  $T$  is an affine monad on  $\mathbf{C}$ , the initial object  $0 \in \mathbf{C}$  is both initial and final in  $\mathcal{Kl}(T')$ . It is always initial, and final since:  $T'(0) = T(0 + 1) \cong T(1) \cong 1$ . Hence 0 is a *zero object* in  $\mathcal{Kl}(T')$ . In particular, for each pair of objects  $X, Y \in \mathbf{C}$  there is

a zero map  $\mathbf{0} = \mathbf{0}_{X,Y}: X \rightarrow T'(Y)$  given by:

$$\mathbf{0}_{X,Y} = \left( X \xrightarrow{!} 1 \cong T'(0) \xrightarrow{T'(!)} T'(Y) \right) = \left( X \xrightarrow{!} 1 \xrightarrow{\eta \circ \kappa_2} T(Y+1) \right)$$

We have  $\mathbf{0} \bullet' f = \mathbf{0} = g \bullet' \mathbf{0}$  for all maps  $f, g$  in  $\mathcal{Kl}(T')$ . We can now define ‘partial projections’  $\triangleright_1: X + Y \rightarrow X$  and  $\triangleright_2: X + Y \rightarrow Y$  in  $\mathcal{Kl}(T')$  via cotuples:

$$\triangleright_1 = \left( X + Y \xrightarrow[\eta \circ (\text{id} + !)]{[\eta \circ \kappa_1, \mathbf{0}]} T(X+1) \right) \quad \triangleright_2 = \left( X + Y \xrightarrow[\eta \circ [\kappa_2 \circ !, \kappa_1]]{[\mathbf{0}, \eta \circ \kappa_1]} T(Y+1) \right).$$

These maps are natural in  $X, Y$ , in the category  $\mathcal{Kl}(T')$ . Notice that  $\triangleright_1: 1 + 1 \rightarrow T(1+1)$  is the unit/identity and  $\triangleright_2: 1 + 1 \rightarrow T(1+1)$  is the swap map  $\eta \circ [\kappa_2, \kappa_1]$ .

We can then form ‘bicartesian’ maps  $\text{bc} = \text{bc}_{X,Y}: T'(X+Y) \rightarrow T'(X) \times T'(Y)$ , as a tuple of the Kleisli liftings of  $\triangleright_1, \triangleright_2$ . That is,

$$\text{bc} = \langle \mu' \circ T'(\triangleright_1), \mu' \circ T'(\triangleright_2) \rangle. \quad (6)$$

**Definition 5.3** [After [7]] An affine monad  $T$  on  $\mathbf{C}$  is *partially additive* if these maps  $\text{bc}$  from (6) are monic in  $\mathbf{C}$ , and the naturality squares below are pullbacks in  $\mathbf{C}$ , for all  $f: X \rightarrow A, g: Y \rightarrow B$  in  $\mathbf{C}$ .

$$\begin{array}{ccc} T'(X+Y) & \xrightarrow{T'(f+g)} & T'(A+B) \\ \text{bc} \downarrow & & \downarrow \text{bc} \\ T'(X) \times T'(Y) & \xrightarrow{T'(f) \times T'(g)} & T'(A) \times T'(B) \end{array} \quad (7)$$

The requirement that  $\text{bc}$  is monic means that the two partial projections  $\triangleright_1: X + Y \rightarrow X, \triangleright_2: X + Y \rightarrow Y$  are jointly monic in  $\mathcal{Kl}(T')$ . In particular, the following two maps in  $\mathcal{Kl}(T)$  are jointly monic (see [8, Assump. 1]).

$$(1+1) + 1 \xrightarrow[\mathcal{W} = [\triangleright_2, \kappa_2] = [[\kappa_2, \kappa_1], \kappa_2]]{\mathcal{W} = [\triangleright_1, \kappa_2] = [\text{id}, \kappa_2]} 1+1 \quad (8)$$

Our next aim is to prove that the Kleisli category  $\mathcal{Kl}(T)$  of a strongly affine partially additive monad  $T$  on a non-trivial distributive category  $\mathbf{C}$  is an effectus.

## 6 The Kleisli category is an effectus

We proceed towards our main theorem in a number of steps, combining the approaches of [7] and [3] (see also [4, Sect. 8]). We first show how to get a FinPAC (after [2]).

**Proposition 6.1** *Let  $T$  be a strongly affine partially additive monad on a non-trivial distributive category  $\mathbf{C}$ . The Kleisli category of the monad  $T' = T((-) + 1)$  is then a finitely partially additive category (a FinPAC, for short, see [2, 4]).*

Explicitly, for maps  $f, g: X \rightarrow T'(Y)$  one says that  $f, g$  are *orthogonal*, written as  $f \perp g$ , if there is a (necessarily unique) bound  $b: X \rightarrow T'(Y + Y)$  such that  $bc \circ b = \langle f, g \rangle$ , *i.e.* such that  $\triangleright_1 \bullet' b = f$  and  $\triangleright_2 \bullet' b = g$ . In that case we define their sum  $\oplus$  by  $f \oplus g = \nabla \bullet' b = T'(\nabla) \circ b: X \rightarrow T'(Y)$ .

The above proposition says that this partial sum  $\oplus$  with the zero map  $\mathbf{0}$  forms a partial commutative monoid (PCM), which is preserved by pre- and post-composition in  $\mathcal{Kl}(T')$  and satisfies the ‘untying axiom’ of [2,12,3]: if  $f \perp g$  then  $(\kappa_1 \bullet' f) \perp (\kappa_2 \bullet' g)$ .

**Proof** All this is rather straightforward and can be copied from [7,4]. We only point out that we need the pullback property (7) in the proof of associativity: let  $f, g, h: X \rightarrow Y$  be given in  $\mathcal{Kl}(T')$  with  $f \perp g$  via bound  $b$ , and  $(f \oplus g) \perp h$  via bound  $c$ . We thus have  $\triangleright_1 \bullet' b = f$ ,  $\triangleright_2 \bullet' b = g$  and  $\triangleright_1 \bullet' c = f \oplus g = \nabla \bullet' b$ ,  $\triangleright_2 \bullet' c = h$ . Consider the following pullback in  $\mathbf{C}$ .

$$\begin{array}{ccc}
 X & \xrightarrow{c} & T'(Y + Y) \\
 \searrow \langle b, h \rangle & \swarrow d & \downarrow bc \\
 T'(Y + Y) \times T'(Y) & \xrightarrow{T'(\nabla + \text{id})} & T'(Y + Y) \\
 & \downarrow bc & \downarrow bc \\
 T'(Y + Y) \times T'(Y) & \xrightarrow{T'(\nabla) \times \text{id}} & T'(Y) \times T'(Y)
 \end{array}$$

Take  $d' = T([[\kappa_2 \circ !, \kappa_1 \circ \kappa_1], \kappa_1 \circ \kappa_2], \kappa_2]) \circ d: X \rightarrow T'(Y + Y)$ . Then  $g \perp h$  via  $d'$ . Next we take  $d'' = T'([\text{id}, \kappa_2]) \circ d: X \rightarrow T'(Y + Y)$ . It proves  $f \perp (g \oplus h)$  and associativity, in:

$$\begin{aligned}
 f \oplus (g \oplus h) &= \nabla \bullet' d'' = T(\nabla + \text{id}) \circ T([\text{id}, \kappa_2] + \text{id}) \circ d \\
 &= T([\nabla, \text{id}] + \text{id}) \circ d \\
 &= T(\nabla + \text{id}) \circ T((\nabla + \text{id}) + \text{id}) \circ d \\
 &= T'(\nabla) \circ T'(\nabla + \text{id}) \circ d \\
 &= \nabla \bullet' c \\
 &= (f \oplus g) \oplus h.
 \end{aligned}$$

The untying axiom follows directly from the way that orthogonality  $\perp$  is defined: if  $f \perp g$ , for  $f, g: X \rightarrow T'(Y)$ , say via bound  $b: X \rightarrow T'(Y + Y)$ , then one can take as new bound  $b' = T'(\kappa_1 + \kappa_2) \circ b: X \rightarrow T'((Y + Y) + (Y + Y))$ . It is easy to see that  $b'$  proves  $(\kappa_1 \bullet' f) \perp (\kappa_2 \bullet' g)$ .  $\square$

The maps  $X \rightarrow 2 = 1 + 1$  in  $\mathcal{Kl}(T)$  are called *predicates* on  $X$ . Equivalently, these predicates may be described as maps  $X \rightarrow T(2)$  in  $\mathbf{C}$ , or as maps  $X \rightarrow 1$  in  $\mathcal{Kl}(T')$ . There are truth and falsity predicates  $\mathbf{1}$  and  $\mathbf{0}$  defined in  $\mathbf{C}$  as:

$$\mathbf{1} = \left( X \xrightarrow{!} 1 \xrightarrow{\kappa_1} 2 \xrightarrow{\eta} T(2) \right) \qquad \mathbf{0} = \left( X \xrightarrow{!} 1 \xrightarrow{\kappa_2} 2 \xrightarrow{\eta} T(2) \right)$$

Orthosupplement is  $p^\perp = T([\kappa_2, \kappa_1]) \circ p$ , so that  $p^{\perp\perp} = p: X \rightarrow T(2)$ . Predicates on 1, of the form  $1 \rightarrow 2$ , are called scalars.

In order to prove that the Kleisli category  $\mathcal{Kl}(T)$  is an effectus the properties below are crucial. They all apply to the associated category  $\mathcal{Kl}(T')$  of partial maps. This emphasis on the partial maps in an effectus is due to [3].

**Lemma 6.2** *For a monad  $T$  as in Proposition 6.1 we additionally have:*

- (i) if  $\mathbf{1} \bullet' f = \mathbf{0}$  then  $f = \mathbf{0}$ , for each  $f: X \rightarrow T(Y + 1)$ ;
- (ii) if  $(\mathbf{1} \bullet' f) \perp (\mathbf{1} \bullet' g)$  then  $f \perp g$ , for all  $f, g: X \rightarrow T(Y + 1)$ ;
- (iii) each homset  $\mathcal{Kl}(T')(X, 1) = \mathcal{Kl}(T)(X, 2) = \mathbf{C}(X, T(2))$  is an effect algebra.
- (iv) if  $T$  is non-trivial, then  $\mathbf{1} \bullet' f = \mathbf{1}$  implies that  $f$  in  $\mathcal{Kl}(T')$  is total, i.e. is of the form  $f = T(\kappa_1) \circ g$ , for a necessarily unique map  $g$  in  $\mathcal{Kl}(T)$ .

**Proof** (i) The assumption  $\mathbf{1} \bullet' f = \mathbf{0}$  mean  $T(! + \text{id}) \circ f = \eta \circ \kappa_2 \circ !$ . Using the pullback in the middle of (5) we obtain  $f = \langle \kappa_2 \rangle \bullet ! = \mathbf{0}$ .

(ii) Let  $(\mathbf{1} \bullet' f) \perp (\mathbf{1} \bullet' g)$ , for  $f, g: X \rightarrow T'(Y)$ , via bound  $b: X \rightarrow T'(1 + 1)$ . Then we use the following pullback instance of (7).

$$\begin{array}{ccccc}
 X & & & & \\
 \searrow c & & \xrightarrow{b} & & \\
 & T'(Y + Y) & \xrightarrow{T'(! + !)} & & T'(1 + 1) \\
 \searrow \langle f, g \rangle & \downarrow \text{bc} \lrcorner & & & \downarrow \text{bc} \\
 & T'(Y) \times T'(Y) & \xrightarrow{T'(!) \times T'(!)} & & T'(1) \times T'(1)
 \end{array}$$

The map  $c$  is by construction a bound for  $f, g$ , showing  $f \perp g$ .

- (iii) Since we already know from Proposition 6.1 that each homset of the Kleisli category  $\mathcal{Kl}(T')$  is a partial commutative monoid (PCM), we only have to prove the following three points.
  - (a) For each predicate  $p$  we have  $p \otimes p^\perp = \mathbf{1}$ .
  - (b) The predicate  $p^\perp$  is unique with this property:  $p \otimes q = \mathbf{1}$  implies  $q = p^\perp$ .
  - (c) If  $\mathbf{1} \perp p$ , then  $p = \mathbf{0}$ .

We shall handle them one by one.

For (a), let  $p: X \rightarrow T(2) = T'(1)$  be a predicate. We take as bound  $b = T(\kappa_1) \circ p: X \rightarrow T'(1 + 1) = T((1 + 1) + 1)$ . One easily checks that  $\triangleright_1 \bullet' b = p$  and  $\triangleright_2 \bullet' b = p^\perp$ , and also that  $p \otimes p^\perp = \nabla \bullet' b = \mathbf{1}$ .

In (b) let  $p \otimes q = \mathbf{1}$ , say via bound  $b: X \rightarrow T'(1 + 1)$ . Then:  $p \otimes q = \mathbf{1} = \nabla \bullet' b = T(\nabla + \text{id}) \circ b = T(! + \text{id}) \circ b$ . The third rectangle in (5) is a pullback in  $\mathcal{Kl}(T)$ , which we use on the left below.

$$\begin{array}{ccc}
 X & \xrightarrow{!} & 1 \\
 \searrow c & & \downarrow \langle \kappa_1 \rangle \\
 & 2 & \xrightarrow{!} 1 \\
 \searrow b & \downarrow \langle \kappa_1 \rangle & \downarrow \langle \kappa_1 \rangle \\
 & 2 + 1 & \xrightarrow{! + \text{id}} 1 + 1
 \end{array}
 \qquad
 \begin{array}{ccc}
 X & \xrightarrow{!} & 1 \\
 \searrow d & & \downarrow \langle \kappa_2 \rangle \\
 & 1 & \xrightarrow{=} 1 \\
 \searrow T(\sigma) \circ b & \downarrow \langle \kappa_2 \rangle & \downarrow \langle \kappa_2 \rangle \\
 & 2 + 1 & \xrightarrow{! + \text{id}} 1 + 1
 \end{array}$$

We thus have  $b = T(\kappa_1) \circ c$ . But then we are done:

$$\begin{aligned}
 p^\perp &= T([\kappa_2, \kappa_1]) \circ p = T([\kappa_2, \kappa_1]) \circ (\triangleright_1 \bullet' b) \\
 &= T([\kappa_2, \kappa_1]) \circ T([\text{id}, \kappa_2]) \circ T(\kappa_1) \circ c \\
 &= T([\kappa_2, \kappa_1], \kappa_2) \circ T(\kappa_1) \circ c \\
 &= \triangleright_2 \bullet' b \\
 &= q.
 \end{aligned}$$

Finally, for (c) let  $\mathbf{1} \perp p$ , say via  $b: X \rightarrow T'(1 + 1)$ , so that  $T([\text{id}, \kappa_2]) \circ b = \triangleright_1 \bullet' b = \mathbf{1} = \langle \kappa_1 \rangle \circ !$ , as in the above diagram on the right. Consider the isomorphism  $\sigma = \mathbb{X} = [[\kappa_2, \kappa_1 \circ \kappa_1], \kappa_1 \circ \kappa_2]: 2 + 1 \xrightarrow{\cong} 2 + 1$ , so that the outer diagram on the right commutes:

$$\begin{aligned}
 T(! + \text{id}) \circ T(\sigma) \circ b &= T([\kappa_2, \kappa_1 \circ ! \circ \kappa_1], \kappa_1 \circ ! \circ \kappa_2) \circ b \\
 &= T([\kappa_2, \kappa_1], \kappa_1) \circ b \\
 &= T([\kappa_2, \kappa_1]) \circ T([\text{id}, \kappa_2]) \circ b \\
 &= T([\kappa_2, \kappa_1]) \circ \langle \kappa_1 \rangle \circ ! \\
 &= \langle \kappa_2 \rangle \circ !.
 \end{aligned}$$

Hence  $T(\sigma) \circ b = \langle \kappa_2 \rangle \circ !$  by the middle pullback in (5). But then:

$$\begin{aligned}
 p &= \triangleright_2 \bullet' b = T([\kappa_2, \kappa_1], \kappa_2) \circ T(\sigma^{-1}) \circ \langle \kappa_2 \rangle \circ ! \\
 &= T([\kappa_2, \kappa_1], \kappa_2) \circ T([\kappa_2 + \text{id}, \kappa_1 \circ \kappa_1]) \circ T(\kappa_2) \circ \eta \circ ! \\
 &= T(\kappa_2) \circ \eta \circ ! \\
 &= \mathbf{0}.
 \end{aligned}$$

- (iv) If  $T$  is a non-trivial monad, then the diagram on the right in (5) is a pullback. Hence the assumption  $\mathbf{1} \bullet' f = \mathbf{1}$  translates to  $(! + \text{id}) \bullet f = \langle \kappa_1 \rangle \bullet !$ , so that there is a unique map  $g$  in  $\mathcal{Kl}(T)$  with  $\langle \kappa_1 \rangle \bullet g = f$ , and thus  $T(\kappa_1) \circ g = f$ .  $\square$

Our main result below gives conditions that ensure that a Kleisli category is an effectus, see [8, 4]. Briefly, an effectus is a category with finite coproducts and a final object in which the two maps  $!V, \mathbb{V}: (1 + 1) + 1 \rightrightarrows 1 + 1$  in (8) are jointly monic, and in which the following diagrams are pullbacks.

$$\begin{array}{ccc}
 X + Y & \xrightarrow{\text{id} + !} & X + 1 \\
 ! + \text{id} \downarrow & & \downarrow ! + \text{id} \\
 1 + Y & \xrightarrow{\text{id} + !} & 1 + 1
 \end{array}
 \qquad
 \begin{array}{ccc}
 X & \xrightarrow{!} & 1 \\
 \langle \kappa_1 \rangle \downarrow & & \downarrow \langle \kappa_1 \rangle \\
 X + Y & \xrightarrow{! + !} & 1 + 1
 \end{array}
 \tag{9}$$

Equivalent conditions can be formulated for the associated category of partial maps, see the original [3, Def. 4.4], copied into [4, Def. 51]. The proof below heavily builds on this partial perspective.

**Theorem 6.3** *A Kleisli category  $\mathcal{Kl}(T)$  is an effectus when  $T$  is a non-trivial strongly affine partially additive monad on a non-trivial distributive category.*

*If the monad  $T$  is additionally commutative, then its Kleisli category  $\mathcal{Kl}(T)$  is a commutative effectus.*

**Proof** Since partial additivity of the monad  $T$  implies that we have jointly monic maps  $(1 + 1) + 1 \rightrightarrows 1 + 1$  in (8), one only has to show that the diagrams in (9) are pullbacks in  $\mathcal{Kl}(T)$ . This is an application of [3, Thm. 4.10], which re-appears as [4, Thm. 53 (2)], using Proposition 6.1 and Lemma 6.2. The category of total maps in  $\mathcal{Kl}(T')$  is then  $\mathcal{Kl}(T)$ , by Lemma 6.2 (iv).

The statement that the Kleisli category  $\mathcal{Kl}(T)$  is a commutative effectus if  $T$  is a commutative monad is based on results (and definitions) from [9].  $\square$

## 7 The monad examples revisited

Our five monad examples  $\mathcal{D}$ ,  $\mathcal{G}$ ,  $\mathcal{V}$ ,  $\mathcal{R}$  and  $\mathcal{E}$  from Section 3 satisfy the assumptions of Theorem 6.3. We concentrate on the probabilistic powerdomain  $\mathcal{V}$  and the Radon monad  $\mathcal{R}$  since the others have been studied elsewhere [9].

### 7.1 The probabilistic powerdomain $\mathcal{V}$

We first check that the probabilistic powerdomain  $\mathcal{V}$  on the category **Cdcpo** of continuous dcpo's is strongly affine. We use the result, due to Lawson, that a valuation on the opens  $\mathcal{O}(X)$  of a continuous dcpo  $X$  can be extended in a unique way to a measure on the Borel sets  $\mathcal{B}(X)$ , see [11,1]. We recall that  $\mathcal{B}(X)$  is the least  $\sigma$ -algebra that contains  $\mathcal{O}(X)$ .

We show that Diagram (4) is a pullback, for  $T = \mathcal{V}$ . The proof is similar to the one for the Giry monad in [9], but uses the unique extension to Borel sets. Let  $\psi \in \mathcal{V}(X \times Y)$  satisfy  $\mathcal{V}(\pi_2)(\psi) = \eta(z)$ , for a given element  $z \in Y$ . This means  $\psi(X \times V) = \psi(\pi_2^{-1}(V)) = \mathcal{V}(\pi_2)(\psi)(V) = \eta(z)(V) = \mathbf{1}_V(z)$ , for each  $V \in \mathcal{O}(Y)$ . We write  $\hat{\psi}: \mathcal{B}(X) \rightarrow [0, 1]$  for the unique extension of  $\psi: \mathcal{O}(X) \rightarrow [0, 1]$ . Since  $\eta$  extends to a measure on  $\mathcal{B}(X)$ , and  $\hat{\psi}(X \times -)$  is also a measure that extends  $\psi(X \times -)$  we get:

$$\hat{\psi}(X \times V) = \mathbf{1}_V(z), \quad \text{for each } V \in \mathcal{B}(X). \quad (10)$$

Our first aim is to show that  $\hat{\psi}$  is non-entwined, that is, satisfies  $\hat{\psi}(U \times V) = \hat{\psi}(U \times Y) \cdot \hat{\psi}(X \times V)$  for all  $U, V \in \mathcal{B}(X)$ . We distinguish two cases.

- If  $z \notin V$ , then by monotonicity:

$$\hat{\psi}(U \times V) \leq \hat{\psi}(X \times V) \stackrel{(10)}{=} \mathbf{1}_V(z) = 0.$$

Hence  $\hat{\psi}(U \times V) = 0 = \hat{\psi}(U \times Y) \cdot \hat{\psi}(X \times V)$ .

- If  $z \in V$ , then  $z \notin \neg V$ . We note that Borel sets (but not open sets) are closed under negation/complement. Hence with the extension  $\hat{\psi}$  to Borel sets we can

reason as follows.

$$\begin{aligned}
\widehat{\psi}(U \times V) &= \widehat{\psi}(U \times V) + 0 \\
&= \widehat{\psi}(U \times V) + \widehat{\psi}(U \times \neg V) \quad \text{as just shown} \\
&= \widehat{\psi}((U \times V) \cup (U \times \neg V)) \quad \text{by additivity} \\
&= \widehat{\psi}(U \times Y) \\
&= \widehat{\psi}(U \times Y) \cdot \mathbf{1}_V(z) \\
&\stackrel{(10)}{=} \widehat{\psi}(U \times Y) \cdot \widehat{\psi}(X \times V).
\end{aligned}$$

But now we are done since we can take  $\phi = \mathcal{V}(\pi_1)(\psi) = \psi(- \times Y) \in \mathcal{V}(X)$ , satisfying:

$$\text{st}_1(\phi, z)(U \times V) = \phi(U) \cdot \mathbf{1}_V(z) \stackrel{(10)}{=} \psi(U \times Y) \cdot \psi(X \times V) = \psi(U \times V).$$

The associated monad  $\mathcal{V}'(X) = \mathcal{V}(X + 1)$  contains sub-valuations  $\phi$ , which need not satisfy  $\phi(X) = 1$ . The map  $\text{bc}: \mathcal{V}'(X + Y) \rightarrow \mathcal{V}'(X) \times \mathcal{V}'(Y)$  from (6) is given by  $\text{bc}(\phi) = \langle \text{bc}_1(\phi), \text{bc}_2(\phi) \rangle$ , where  $\text{bc}_i(\phi)(U) = \phi(\kappa_i U)$ . This map is clearly injective. We leave it to the reader to verify that the naturality squares are pullbacks.

## 7.2 The Radon monad $\mathcal{R}$

The proof that the Radon monad is strongly affine that is presented below is due to Robert Furber; it is analogous to the proof for  $\mathcal{V}$ , but uses the Cauchy-Schwartz inequality for positive maps on  $C^*$ -algebras. We first note that the strength map  $\text{st}_1: \mathcal{R}(X) \times Y \rightarrow \mathcal{R}(X \times Y)$  is determined by  $\text{st}_1(\omega, z)(\phi \otimes \psi) = \omega(\phi) \cdot \psi(z)$ . These tensors  $\phi \otimes \psi = \lambda(x, y) \cdot \phi(x) \cdot \psi(y) \in C(X \times Y) \cong C(X) \otimes C(Y)$  form a dense subset. Hence the above description of  $\text{st}_1$  suffices.

We turn to Diagram (4). Let  $\omega \in \mathcal{R}(X \times Y)$  and  $z \in Y$  be given with  $\mathcal{R}(\pi_2)(\omega) = \eta(z)$ . This means that  $\omega(\mathbf{1} \otimes \psi) = \psi(z)$ , for each  $\psi \in C(Y)$ , where  $\mathbf{1} \in C(X)$  is the function that is constantly 1. The Cauchy-Schwartz inequality for the positive map  $\omega$  yields:

$$\begin{aligned}
|\omega(\phi \otimes \psi)|^2 &= \omega((\phi \otimes \mathbf{1}) \cdot (\mathbf{1} \otimes \psi))^* \cdot \omega((\phi \otimes \mathbf{1}) \cdot (\mathbf{1} \otimes \psi)) \\
&\leq \omega((\phi \otimes \mathbf{1}) \cdot (\phi \otimes \mathbf{1})^*) \cdot \omega((\mathbf{1} \otimes \psi)^* \cdot (\mathbf{1} \otimes \psi)) \\
&= \omega((\phi \cdot \phi^*) \otimes \mathbf{1}) \cdot \omega(\mathbf{1} \otimes (\psi^* \cdot \psi)) \\
&= \omega((\phi \cdot \phi^*) \otimes \mathbf{1}) \cdot (\psi^* \cdot \psi)(z) \\
&= \omega((\phi \cdot \phi^*) \otimes \mathbf{1}) \cdot \psi(z)^* \cdot \psi(z).
\end{aligned}$$

Hence if  $\psi(z) = 0$ , then  $\omega(\phi \otimes \psi) = 0$ . Consider the function  $\psi' \in C(Y)$  given by  $\psi'(y) = \psi(z) - \psi(y)$ . Since  $\psi'(z) = 0$ , we get  $\omega(\phi \otimes \psi') = 0$ , as just shown, and



thus by linearity of  $\omega$ :

$$\begin{aligned}
\omega(\phi \otimes \psi) &= \omega(\phi \otimes \psi) + \omega(\phi \otimes \psi') = \omega(\phi \otimes (\psi - \psi')) \\
&= \omega(\phi \otimes \psi(z)) \\
&= \omega(\phi \otimes \mathbf{1}) \cdot \psi(z) \\
&= \omega(\phi \otimes \mathbf{1}) \cdot \omega(\mathbf{1} \otimes \psi).
\end{aligned}$$

We can now take as state  $\rho = \mathcal{R}(\pi_1)(\omega) \in \mathcal{R}(X)$  given by  $\rho(\phi) = \omega(\phi \otimes \mathbf{1})$ . This gives the mediating element we seek, since:

$$\text{st}_1(\rho, z)(\phi \otimes \psi) = \rho(\phi) \cdot \psi(z) = \omega(\phi \otimes \mathbf{1}) \cdot \omega(\mathbf{1} \otimes \psi) = \omega(\phi \otimes \psi).$$

The monad  $\mathcal{R}'(X) = \mathcal{R}(X+1)$  contains the states on  $C(X+1) \cong C(X) \oplus \mathbb{C}$ , and thus the subunital positive maps  $C(X) \rightarrow \mathbb{C}$ , which are also known as substates. The map  $\text{bc}: \mathcal{R}'(X+Y) \rightarrow \mathcal{R}'(X) \times \mathcal{R}'(Y)$  is given by  $\text{bc}(\omega) = (\omega_1, \omega_2)$ , where  $\omega_1(\phi) = \omega([\phi, \mathbf{0}])$  and  $\omega_2(\psi) = \omega([\mathbf{0}, \psi])$ . It is obviously injective.

## 8 Conclusions and outlook

Our main result gives sufficient conditions for a monad so that its Kleisli category is an effectus. These conditions are, roughly: strong affineness and partial additivity. This solves a problem that has been open for a couple of years, since the inception of effectus theory. In [9] it is shown that strong affineness of a monad  $T$  gives a bijective correspondence between predicates  $X \rightarrow 2$  in  $\mathcal{Kl}(T)$  and instruments  $f: X \rightarrow X+X$  in  $\mathcal{Kl}(T)$  which are side-effect-free, in the sense that  $\nabla \bullet f = \text{id}$ . This part of the definition of a commutativity effectus. In [4, Example 58] one more property is used that is important for probabilistic computation, namely normalisation, giving conditional probability.

We expect that commutativity and normalisation play a role in a categorical characterisation of probabilistic computation that we have as long term goal, as discussed in the introduction to this paper.

### Acknowledgements

Thanks to Kenta Cho, Robert Furber, and Bram Westerbaan for helpful discussions, and to the anonymous referees for their critical feedback.

## References

- [1] M. Alvarez-Manilla, A. Edalat, and N. Saheb-Djahromi. An extension result for continuous valuations. *Journ. London Math. Soc.*, 61(02):629–640, 2000.
- [2] M. Arbib and E. Manes. Partially additive categories and flow-diagram semantics. *Journ. Algebra*, 62(1):203–227, 1980.
- [3] K. Cho. Total and partial computation in categorical quantum foundations. In C. Heunen, P. Selinger, and J. Vicary, editors, *Quantum Physics and Logic (QPL) 2015*, number 195 in Elect. Proc. in Theor. Comp. Sci., pages 116–135, 2015.
- [4] K. Cho, B. Jacobs, A. Westerbaan, and B. Westerbaan. An introduction to effectus theory. see [arxiv.org/abs/1512.05813](https://arxiv.org/abs/1512.05813), 2015.

- [5] R. Cockett. Introduction to distributive categories. *Math. Struct. in Comp. Sci.*, 3:277–307, 1993.
- [6] R. Furber and B. Jacobs. From Kleisli categories to commutative  $C^*$ -algebras: Probabilistic Gelfand duality. *Logical Methods in Comp. Sci.*, 11(2):1–28, 2015.
- [7] B. Jacobs. From coalgebraic to monoidal traces. In B. Jacobs, M. Niqui, J. Rutten, and A. Silva, editors, *Coalgebraic Methods in Computer Science*, volume 264 of *Elect. Notes in Theor. Comp. Sci.*, pages 125–140. Elsevier, Amsterdam, 2010.
- [8] B. Jacobs. New directions in categorical logic, for classical, probabilistic and quantum logic. *Logical Methods in Comp. Sci.*, 11(3):1–76, 2015.
- [9] B. Jacobs. Affine monads and side-effect-freeness. CMCS’16, 2016.
- [10] C. Jones. *Probabilistic Non-determinism*. PhD thesis, Edinburgh Univ., 1989.
- [11] C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In *Logic in Computer Science*, pages 186–195. IEEE, Computer Science Press, 1989.
- [12] E. Manes and M. Arbib. *Algebraic Approaches to Program Semantics*. Texts and Monogr. in Comp. Sci., Springer, Berlin, 1986.

# Programs as Data Structures in $\lambda SF$ -Calculus

Barry Jay

*Centre for Quantum Computing & Intelligent Systems  
School of Software  
University of Technology Sydney  
Sydney  
Australia*

---

## Abstract

Lambda-SF-calculus can represent programs as closed normal forms. In turn, all closed normal forms are data structures, in the sense that their internal structure is accessible through queries defined in the calculus, even to the point of constructing the Goedel number of a program. Thus, program analysis and optimisation can be performed entirely within the calculus, without requiring any meta-level process of quotation to produce a data structure.

Lambda-SF-calculus is a confluent, applicative rewriting system derived from lambda-calculus, and the combinatory SF-calculus. Its superior expressive power relative to lambda-calculus is demonstrated by the ability to decide if two programs are syntactically equal, or to determine if a program uses its input. Indeed, there is no homomorphism of applicative rewriting systems from lambda-SF-calculus to lambda-calculus. Program analysis and optimisation can be illustrated by considering the conversion of a programs to combinators. Traditionally, a program  $p$  is interpreted using fixpoint constructions that do not have normal forms, but combinatory techniques can be used to block reduction until the program arguments are given. That is,  $p$  is interpreted by a closed normal form  $M$ . Then factorisation (by  $F$ ) adapts the traditional account of lambda-abstraction in combinatory logic to convert  $M$  to a combinator  $N$  that is equivalent to  $M$  in the following two senses. First,  $N$  is extensionally equivalent to  $M$  where extensional equivalence is defined in terms of eta-reduction. Second, the conversion is an intensional equivalence in that it does not lose any information, and so can be reversed by another definable conversion. Further, the standard optimisations of the conversion process are all definable within lambda-SF-calculus, even those involving free variable analysis.

Proofs of all theorems in the paper have been verified using the Coq theorem prover.

*Keywords:* lambda-calculus, SF-calculus, self-interpretation, xi-rule

---

## 1 Introduction

$\lambda$ -calculus [1] provides a completely general account of the extensional behaviour of functions, of all that can be discovered by evaluating them. This may be enough for applications, but the implementation of programming languages requires access to the internal structure of programs. As this is not possible from within the pure  $\lambda$ -calculus, meta-level analysis is commonly required. For example, self-interpretation of  $\lambda$ -calculus [12,17,2,14,15,4,3,18,9], usually begins by applying a meta-function

---

<sup>1</sup> Thanks to Thomas Given-Wilson, Neil Jones, Jens Palsberg and Jose Vergara for helpful discussions as this work gestated, and the anonymous referees.

<sup>2</sup> Email: [Barry.Jay@uts.edu.au](mailto:Barry.Jay@uts.edu.au)

$$\begin{aligned}
M, N, P &:= x \mid S \mid F \mid \lambda x.M \mid MN \\
(\lambda x.M)N &\longrightarrow \{N/x\}M \\
SMNP &\longrightarrow MP(NP) \\
FOMN &\longrightarrow M \quad (O \text{ is } S \text{ or } F) \\
FPMN &\longrightarrow NP \mid [P] \quad (P \text{ is a compound of } P \mid \text{ and } [P]).
\end{aligned}$$

Fig. 1.  $\lambda SF$ -calculus

**quote** which converts an arbitrary  $\lambda$ -term into a data structure, whose internal structure can be queried at will.

Recent work suggests an alternative approach, using calculi that support a more general class of queries. *Pure pattern calculus* [8,5] uses pattern matching to define *generic queries* of data structures built from arbitrary constructors. However, it is unable to analyse pattern-matching functions themselves. *SF-calculus* [7] can query any closed normal form by using its operator  $F$  to reveal its internal structure, e.g. the components  $P_1$  and  $P_2$  of a closed normal application  $P_1 P_2$ . However, it does not provide first-class support for  $\lambda$ -abstraction or any other mechanism for binding variables.

This paper shows how to factorise abstractions in a new calculus, the  $\lambda SF$ -calculus, by converting them to combinators when it is safe to do so, i.e. when this will not break any redexes in the body of the abstraction. The syntax and reduction rules of  $\lambda SF$ -calculus are just those of  $\lambda$ -calculus and  $SF$ -calculus, as given in Figure 1, on the understanding that the compounds now include some abstractions as well as some applications. The result is a proper extension of  $\lambda$ -calculus in the sense that there is *no* function from  $\lambda SF$ -calculus to  $\lambda$ -calculus that preserves its structure as an applicative rewriting system.

This expressive power can be used to support arbitrary queries of closed normal forms. In this sense, we can identify the *data structures* with the closed normal forms. What about programs? The standard interpretation of programs does *not* yield normal forms since recursion is modeled by a fixpoint function that does not have a normal form. However, traditional combinators can be used to identify programs, even recursive ones, with closed normal forms. Hence, we can identify the programs with closed normal forms, to get

$$\text{programs} = \text{closed normal forms} = \text{data structures}.$$

That is, programs can be represented by terms that are simultaneously functions, ready to act on arguments, and data structures, ready for analysis and optimisation, and this without any need for quotation. Except when justifying this equation, we will identify the programs with the closed normal forms.

This provides a more flexible foundation for computation than any of the traditional models, as these emphasise only one aspect of a program's nature. In particular,  $\lambda$ -calculus emphasises its functional aspect, while Turing machines emphasise its structure, as a string of symbols on a tape. This new flexibility suggests fresh approaches to many issues in theory and practice, especially the implementation of

programming languages.

The structure of the paper is as follows. Section 1 is the introduction. Section 2 introduces  $\lambda SF$ -calculus and its basic properties. Section 3 shows that equality of programs is definable. Section 4 defines extensional equivalence. Section 5 shows that there is no homomorphism of applicative rewriting systems from  $\lambda SF$ -calculus to  $\lambda$ -calculus. Section 6 show how to represent recursive programs as closed normal forms. Section 7 converts programs to extensionally equivalent combinators. Section 8 optimises the conversion by program analysis. Section 9 converts programs to combinators in a way that preserves intensions as well as extensions. Section 9 discusses the proof verifications in Coq. Section 10 suggests some fresh approaches to existing issues. Section 11 draws conclusions.

## 2 $\lambda SF$ -calculus

The terms and reduction rules of  $\lambda SF$ -calculus are given in Figure 1. The terms (meta-variables  $M, N, P, \dots$  consist of *variables*  $x, y, z, \dots, f, g, \dots$ , the *operators*  $S$  and  $F$ , *abstractions*  $\lambda x.M$  with bound variable  $x$  and *body*  $M$ , and *applications*  $MN$  of  $M$  to  $N$ . The reduction rules for  $\lambda$  and  $S$  are standard. The rules for  $F$  have the same high-level semantics as in  $SF$ -calculus in that  $F$  branches according to whether its first argument  $P$  is an *atom*, i.e. an operator, or a *compound*. If  $P$  is an atom then return the first branch: if  $P$  is a compound then apply the second branch to its two components. The intention is that the compounds are terms whose decomposition into components does not break any redexes. They are, in a sense, head normal forms. The technical point is that there is a syntactic test for this property, even in the presence of abstractions. The reflexive, transitive closure of  $\longrightarrow$  is denoted  $\longrightarrow^*$ .

### 2.1 Compounds

In combinatory calculi, the compounds are all the partially applied operators. For example, in  $SF$ -calculus, the compounds are all terms of the form  $SM$  or  $SMN$  or  $FM$  or  $FMN$ . These forms are compounds in  $\lambda SF$ -calculus, too. All other compounds of  $\lambda SF$ -calculus are abstractions  $\lambda x.M$  whose decomposition is safe because either  $M$  is already an atom or compound, or outermost reduction in  $M$  awaits the instantiation of  $x$ , i.e.  $x$  is *active* in  $M$  in the following sense.

Define the set  $\text{active}(M)$  of *active variables* of a term  $M$  to be a set that has at most one element, that is defined by the pattern-matching function in Figure 2.1 ( $\text{active}(M) - \{x\}$  removes  $x$  from  $\text{active}(M)$ ).

Here are some examples of compounds. The body of  $\lambda x.x y$  has  $x$  active. The body of  $\lambda x.\lambda y.x$  has  $x$  active. The body of  $\lambda x.\lambda y.y$  is a compound. The body of  $\lambda x.F$  is an atom. The body of  $\lambda x.Fx$  is a compound. The body of  $\lambda x.FxM$  is a compound. The body of  $\lambda x.FxMN$  has  $x$  active, since  $F$  is an intensional operator that needs to know the value of  $x$  to reduce. The body of  $\lambda x.\lambda y.F(FxMN)PQ$  has  $x$  active.

```

active =
|  $x \Rightarrow \{x\}$ 
|  $O \Rightarrow \{\}$ 
|  $\lambda x.M \Rightarrow \text{active}(M) - \{x\}$ 
|  $OM \Rightarrow \{\}$ 
|  $OMN \Rightarrow \{\}$ 
|  $SMNP \Rightarrow \{\}$ 
|  $FMNP \Rightarrow \text{active } M$ 
|  $MN \Rightarrow \text{active } M \quad \text{otherwise.}$ 

```

Fig. 2. Active Variables

## 2.2 Star Abstraction

The decomposition of an abstraction  $\lambda x.M$  will use the *star abstraction*  $\lambda^*x.M$  of  $M$  with respect to  $x$ . This is an adaptation of the standard technique for defining the abstraction of a combinator  $M$  with respect to a variable. Since this is defined using the combinators  $S, K$  and  $I$ , the latter two must be defined in terms of  $S$  and  $F$ , as follows. Define

$$K = FF$$

so that  $KMN = FFMN \longrightarrow M$  for any choice of  $M$  and  $N$ . Then define

$$I = SKK$$

so that  $IM = SKKM \longrightarrow KM(KM) \longrightarrow M$  for any  $M$ .

The *star abstraction*  $\lambda^*x.M$  of  $M$  with respect to  $x$  is defined by

$$\begin{aligned}
\lambda^*x.x &= I \\
\lambda^*x.y &= Ky \quad (y \neq x) \\
\lambda^*x.O &= KO \quad (O \text{ an operator}) \\
\lambda^*x.\lambda y.M &= \lambda x.\lambda^*y.M \\
\lambda^*x.MN &= S(\lambda x.M)(\lambda x.N) .
\end{aligned}$$

This definition modifies the traditional definition of  $\lambda^*x.M$  for combinators  $M$  in two ways. First, when the body is an application  $MN$  the result uses  $\lambda x.N$  instead of  $\lambda^*x.N$ . To see why this is necessary, consider  $\lambda^*x.F(KN_1N_2)$ . Now  $F(KN_1N_2)$  is a compound, so it is safe to separate  $F$  from  $KN_1N_2$  but  $\lambda^*x.KN_1N_2$  breaks the redex  $KN_1N_2$  so a recursive call to  $\lambda^*x$  would here be unsafe. Second, there needs to be a rule for  $\lambda^*x$  when the body is an abstraction  $\lambda y.M$ . The result is  $\lambda x.\lambda^*y.M$  and not  $\lambda^*x.\lambda^*y.M$  since it is important that only one abstraction is eliminated at a time, namely, the innermost one.

Here are some simple examples of star abstraction. In  $SKI$ -calculus, the  $\lambda$ -abstraction  $\lambda x.\lambda y.y$  can be represented by

$$\lambda^*x.\lambda^*y.y = \lambda^*x.I = KI$$

where  $\lambda^*$  is used to convert abstractions into combinators in the traditional manner. In  $\lambda SF$ -calculus, the  $\lambda x.\lambda y.y$  is already a closed normal form. However, its factorisation will introduce  $\lambda^*x.\lambda y.y$  which is calculated as follows:

$$\lambda^*x.\lambda y.y = \lambda x.\lambda^*y.y = \lambda x.I .$$

This has eliminated the innermost abstraction, just like the first step in the calcu-

lation of  $\lambda^*x.\lambda^*y.y$  in *SKI*-calculus. A second factorisation exposes

$$\lambda^*x.SKK = S(\lambda x.SK)(\lambda x.K) .$$

Further factorisation eliminates the remaining abstractions to produce the combinator

$$S(S(KS)(S(KF)(KF)))(S(KF)(KF))$$

which when applied to terms  $M$  and  $N$  reduces to  $N$ , just like the original abstraction. Of course, it is much bigger than the original term, as it does not take advantage of the standard optimisation, in which  $\lambda^*x.I$  takes advantage of the fact that  $x$  is not free in  $I$  to produce  $KI$ . This will be addressed in Section 8.

### 2.3 Components

The *left component*  $M]$  of a term  $M$  is defined as follows

$$\begin{aligned} (MN)] &= M \\ M] &= \text{abs\_left} \quad (\text{otherwise}) \end{aligned}$$

where  $\text{abs\_left} = SKF$  will be used as the left component of any term that is not an application, especially of any abstraction. The key point about  $\text{abs\_left}$  is that it cannot be the left component of an application to some  $N$  since  $\text{abs\_left } N = SKFN$  is a fully applied instance of  $S$ . In general, words in sans-serif, such a  $\text{abs\_left}$  may be used to name particular terms of  $\lambda SF$ -calculus, as well as the meta-variables  $M$  and  $N$ , etc.

Now the *right component*  $\lceil M$  of  $M$  is defined by

$$\begin{aligned} \lceil (MN) &= N \\ \lceil (\lambda x.M) &= \lambda^*x.M \\ \lceil M &= M \quad (\text{otherwise.}) \end{aligned}$$

It follows that if  $M$  is a compound and  $M \longrightarrow N$  then  $M] \longrightarrow N]$  and  $\lceil M \longrightarrow \lceil N$ . That is, no redexes are broken by taking components of compounds. To put it another way, there is a derived reduction rule

$$(\xi) \frac{M \longrightarrow N}{\lambda^*x.M \longrightarrow \lambda^*x.N} \quad (\lambda x.M \text{ is a compound.})$$

### 2.4 Confluence

**Theorem 2.1 (confluence\_lamSF\_red)** *Reduction in  $\lambda SF$ -calculus is confluent.*

**Proof.** The proof can be seen as an instantiation of Klop's result [13] for extensions of  $\lambda$ -calculus, in that the additional reduction rules are left-linear and orthogonal. The only catch is that the reduction rule for  $F$  has a side-condition, so some care is required.  $\square$

### 2.5 Normal Forms

The *normal forms* are defined to be the variables, operators, abstractions of normal forms, and applications  $MN$  in which  $M$  and  $N$  are both normal and  $MN$  is either

a compound or has an active variable.

**Theorem 2.2 (irreducible\_iff\_normal)** *A term is irreducible if and only if it is a normal form.*

A *program* is a closed normal form. A *factorable form* is either an operator or a compound.

**Theorem 2.3 (programs\_are\_factorable)** *All programs are factorable forms.*

Hence, any closed term of the form  $FPMN$  must reduce. This is a form of progress result.

### 3 Definable Equality

It follows from Theorem 2.3 that the equality term defined in  $SF$ -calculus [7] serves to define equality in  $\lambda SF$ -calculus too. The algorithm is as follows. Operators are equal if they have the same extensional behaviour, which can be decided by some term `eqop`. Atoms and compounds are never equal. Compounds are equal if their components are. The actual term is given

$$\text{fix } (\lambda e. \lambda x. \lambda y. F x (\text{eqop } x y) (\lambda x_l. \lambda x_r. F y (KI) (\lambda y_l. \lambda y_r. e x_l y_l (e x_r y_r) (KI)))) .$$

where `fix` is a fixpoint term. This, and other approaches to recursion, will be addressed in Section 6.

**Theorem 3.1 (equal\_programs)** *equal  $M M \longrightarrow^* K$  for all programs  $M$ .*

**Theorem 3.2 (unequal\_programs)** *equal  $M N \longrightarrow^* KI$  for all distinct programs  $M$  and  $N$ .*

**Proof.** The proof is by induction on the rank of  $M$ , as defined in the Coq implementation. The only case of interest arises when  $M$  is an abstraction and  $N$  is an application. Now the left component of  $N$  cannot be `abs_left` since any application of `abs_left` reduces, and so the left components of  $M$  and  $N$  cannot be equal.  $\square$

### 4 Extensionality

Mathematically, two functions  $f$  and  $g$  are extensionally equivalent if they have the same graph. For unary functions, this means that  $f x = g x$  for all  $x$ . In  $\lambda$ -calculus, extensionality is captured by adding the  $\eta$ -reduction rule

$$\lambda x. f x \longrightarrow f \quad \text{if } x \text{ is not free in } f.$$

When added to the basic  $\lambda$ -calculus, with just the  $\beta$ -rule, we get the  $\lambda\beta\eta$ -calculus, which is confluent. Define  $=_{\beta\eta}$  to be the equivalence relation on  $\lambda$ -calculus induced by  $\beta$ -reduction and  $\eta$ -reduction. However, adding the  $\eta$ -rule to  $\lambda SF$ -calculus is unsound, as can be seen from the following calculations. Define  $\equiv_{\beta\eta SF}$  to be the equivalence relation on  $\lambda SF$  induced by its reduction rules and the  $\eta$ -rule. First, the operators  $S, K$  and  $I$  become equal to their usual interpretations, by

$$\begin{aligned} S &\equiv_{\beta\eta SF} \lambda x. \lambda y. \lambda z. Sxyz \equiv_{\beta\eta} \lambda x. \lambda y. \lambda z. xz(yz) \\ K &\equiv_{\beta\eta SF} \lambda x. \lambda y. Kxy \equiv_{\beta\eta SF} \lambda x. \lambda y. x \end{aligned}$$



$$I \equiv_{\beta\eta SF} \lambda x. Ix \equiv_{\beta\eta SF} \lambda x. x .$$

Then we have  $SKM \equiv_{\beta\eta SF} \lambda x. x \equiv_{\beta\eta SF} (SKN)$  for any terms  $M$  and  $N$ . Further,

$$F(SK M)I(KI) \equiv_{\beta\eta SF} KI(SK)M \equiv_{\beta\eta SF} M$$

shows that  $M \equiv_{\beta\eta SF} N$  and this for any  $M$  and  $N$ . The calculus has collapsed.

A more useful relation is obtained by excluding the rule for factoring compounds from the equivalence relation, to get the equivalence relation  $\equiv_{\beta\eta SK}$ . Define terms  $M$  and  $N$  of  $\lambda SF$  to be *extensionally equivalent* if  $M \equiv_{\beta\eta SK} N$ . For example, we have the following lemma.

**Lemma 4.1 (star\_equiv\_abs)**  $\lambda^*x.M \equiv_{\beta\eta SK} \lambda x.M$  for all terms  $M$ .

Here are three more examples of definable program manipulations that preserve extensional behaviour.

Define a combinator **wait** so that

$$\text{wait } M \ N \longrightarrow^* S(S(KM)(KN))I$$

using standard combinatorial techniques. The right-hand side is normal if  $M$  and  $N$  are, but application to some  $P$  reduces this to  $M \ N \ P$  so that **wait**  $M \ N$  waits for  $P$  before applying  $M$  to  $N$ . It follows that

**Lemma 4.2 (wait\_ext)** For all terms  $M$  and  $N$ ,  $\text{wait } M \ N \equiv_{\beta\eta SK} M \ N$ .

Define a combinator **tag** with the property that

$$\text{tag } T \ M \longrightarrow^* S(KM)(SKT) .$$

Now  $SKT$  is an identity function for any  $T$  since

$$SKTP \longrightarrow KP(TP) \longrightarrow P .$$

It follows that when **tag**  $M \ N$  is applied to some  $P$  then it reduces by

$$S(KM)(SKT)P \longrightarrow KMP(SKTP) \longrightarrow^* MP .$$

**Lemma 4.3 (tag\_ext)** For all terms  $T$  and  $M$ ,  $\text{tag } T \ M \equiv_{\beta\eta SK} M$ .

The resulting system of tags is as rich as the calculus as a whole, and so can be used to carry information about, say, constructors or types. In this paper, we will use just three tags in program analysis as follows: **abs** = **tag**  $F$  will tag abstractions; **com** = **tag**  $S$  will tag combinators; and **app** =  $\lambda x. \lambda y. \text{tag } K \ (\text{wait } x \ y)$  will tag applications.

Define a combinator **eager** such that **eager**  $M \ N$  reduces to  $M \ N$  if and only if  $N$  is factorable. That is, replacing  $M \ N$  by **eager**  $M \ N$  forces the application to evaluate  $N$  before evaluating the application of  $M$  to it. The target is a term similar to

$$FN(\lambda x. xN)(\lambda y. \lambda z. \lambda x. x(yz))M$$

where  $x, y$  and  $z$  are fresh. Now  $M$  is applied to  $N$  only if  $N$  has been reduced to factorable form. The new abstractions can be eliminated by ensuring

$$\text{eager } M \ N \longrightarrow^* FN(SI(KN))(S(K(S(K(SI))))(S(KK)))$$

This will be used later to block non-terminating reductions. However, if your goal is to avoid re-computation of  $N$  then this approach has the weakness that

if  $N$  reduces to an operator then it will be evaluated twice! This problem can be eliminated by introducing a variant of  $F$  in which the atomic branch uses its argument.

**Lemma 4.4 (eager\_is\_eager)**  $\text{eager } M N \longrightarrow^* M N$  for all programs  $M$  and  $N$ .

## 5 Homomorphisms

The common features shared by all these calculi are that they are *applicative rewriting systems* [20] that have variables as a term form. Accordingly, it makes sense to define a *homomorphism* of applicative rewriting systems with variables to be a function from one such to another which has the following characteristics:

- it preserves the equivalence relation derived from reduction;
- it preserves applications;
- it preserves variables;
- it does not introduce free variables.

It is enough to require preservation up to equivalence, but for convenience, we will demand strict equality. Similarly, the requirement that a homomorphism does not introduce free variables can be weakened to require that closed terms be mapped to closed terms, or even that operators be mapped to closed terms. Note that the definition does *not* require that  $\lambda$ -abstractions be preserved, or that the image of  $S$  takes any particular form. All conditions are expressed in terms of concepts common to all the calculi under consideration, namely rewriting, variables and applications.

**Theorem 5.1 (no\_homomorphism)** *There is no homomorphism from  $\lambda SF$ -calculus to  $\lambda$ -calculus.*

**Proof.** Assume that there is such a homomorphism. Then it can be composed with the embedding of  $\lambda$ -calculus into  $\lambda\beta\eta$ -calculus to get a homomorphism  $[-]$ . It follows that

$$[S] \equiv_{\beta\eta} \lambda x. \lambda y. \lambda z. [S]xyz \equiv_{\beta\eta} \lambda x. \lambda y. \lambda z. [Sxyz] \equiv_{\beta\eta} \lambda x. \lambda y. \lambda z. xz(yz) .$$

Similarly, we can show that  $[K] \equiv_{\beta\eta} \lambda x. \lambda y. x$  and  $[I] \equiv_{\beta\eta}$ -equivalent to  $\lambda x. x$ . Finally, in  $SF$ -calculus we have  $F(SKM)F(KI) \longrightarrow KI(SK)M \longrightarrow^* M$ , for each term  $M$ , and so

$$\begin{aligned} [F(SKM)I(KI)] &\equiv_{\beta\eta} [F]([S][K][M])[I]([K][I]) \\ &\equiv_{\beta\eta} [F](\lambda x. x)(\lambda x. x)(\lambda x. \lambda y. y) . \end{aligned}$$

Hence, by the homomorphism property, we have

$$[M] \equiv_{\beta\eta} [F](\lambda x. x)(\lambda x. x)(\lambda x. \lambda y. y) .$$

Now the right-hand side is independent of  $M$  and so we have, for any  $N$ , that  $[M] \equiv_{\beta\eta} [N]$ . In particular we have  $x = [x] \equiv_{\beta\eta} [y] = y$  for any variables  $x$  and  $y$ , which yields a contradiction.  $\square$

**Corollary 5.2** *There is no homomorphism from  $SF$ -calculus to  $\lambda$ -calculus.*

**Proof.** The proof of Theorem 5.1 applies equally to  $SF$ -calculus.  $\square$

## 6 Programs as Normal Forms

The identification of programs with (closed) normal forms in an untyped setting is rather unusual. Of course, we cannot isolate the terminating computations, as this would solve the Halting Problem. If, further, we allow any computation to be a program, i.e. albeit one that takes no inputs, then the game is over. However, by separating the program from its inputs, we can use combinatory techniques to block any troublesome reductions in the program until the input is given. In this manner, programs can be made strongly normalising, and so can be identified with (closed) normal forms.

A crude solution would be to replace all abstractions with the corresponding star abstractions, as these are closed normal forms by construction. However, this is surely more violent than necessary.

Ideally, the identification should be demonstrated using a small programming language, with a conversion function from programs to closed normal forms of  $\lambda SF$ -calculus, but this is beyond our current scope. There may be several ways to do this, and the options will change dramatically if the language is typed. Rather than explore these options, which would take some time, let us rather show how to overcome the key difficulty, namely the representation of recursive programs.

Consider a recursive program of the form

$$\text{let rec } f \text{ } x = M$$

where  $M$  may contain  $f$  and  $x$  as free variables. Its standard representation is by a term of the form  $\text{fix } (\lambda f. \lambda x. M)$  where  $\text{fix}$  is a fixpoint function defined to be  $\omega\omega$  where  $\omega = \lambda x. \lambda f. f(xxf)$ . It follows that

$$\text{fix} = \lambda x. \lambda f. f(xxf)\omega \longrightarrow \lambda f. f(\omega\omega f) = \lambda f. f(\text{fix } f) .$$

so that  $\text{fix } f \longrightarrow f (\text{fix } f)$ . This expresses the recursion very cleanly, but now program representations do not have a normal form.

However, we can delay the application of  $\omega$  to  $\omega$  by replacing  $\text{fix}$  by the extensionally equivalent term

$$\text{fix2} = \lambda f. \text{wait } (\text{wait } \omega \omega) f .$$

Its application to a normal form  $f$  also has a normal form, but further application to some  $x$  reduces to  $\omega \omega f x$  which is  $\text{fix } f x$ . Now the original program can be interpreted by  $\text{fix2 } (\lambda f. \lambda x. M)$ . In the same manner, we may define  $\text{fix3}$  and  $\text{fix4}$  etc, so that recursive programs can be made to wait for any number of arguments before risking non-termination.

This accounts for the outermost recursion in a program, but when recursive functions are composed then this technique produces terms of the form

$$\lambda x. \text{fix2 } f (\text{fix2 } g x)$$

which re-introduces arbitrary computations into programs through  $\text{fix2 } g x$ . To block this, introduce eager evaluation, as described in Section 4 and define yet another fixpoint term by

$$\text{fix.eager} = \lambda f. \lambda x. \text{eager } (\text{wait } (\text{wait } \omega \omega) f) x .$$

Now the composition of recursive programs normalises since evaluation of the recursion is blocked until the bound variable  $x$  takes a value. In this manner, the core

constructions used to create recursive programs can be controlled by combinators to ensure that they do not introduce non-termination.

## 7 Extensional Conversion to Combinators

The extensional conversion of program to combinators is given by the recursive, pattern-matching function

```
to_combinator :=
| O ⇒ O
| λx.M ⇒ to_combinator (λ*x.M)
| MN ⇒ (to_combinator M) (to_combinator N))
```

which eventually converts each abstraction  $\lambda x.M$  in its argument to  $\lambda^*x.M$ .

**Theorem 7.1 (to\_combinator\_makes\_combinators)** *If  $M$  is a closed term then  $\text{to\_combinator } M$  is a combinator.*

Since it is easy to test for abstractions and compounds, there is no difficulty in representing  $\text{to\_combinator}$  as a program, namely,

$$\text{to\_comb} = \text{fix}(\lambda f. \lambda x. F \ x \ x \ (\lambda x_l. \lambda x_r. \text{equal} \ \text{abs\_left} \ x_l \ (f \ x_r) \ ((f \ x_l) \ (f \ x_r))))).$$

**Theorem 7.2 (to\_combinator\_is\_extensional)**  $\text{to\_combinator } M \equiv_{\beta\eta SK} M$  for all terms  $M$ .

**Theorem 7.3 (to\_combinator\_to\_comb)** *For all programs  $M$  we have*

$$\text{to\_comb } M \longrightarrow^* \text{to\_combinator } M.$$

Summarising, if  $M$  is a program then  $\text{to\_comb } M$  reduces to the combinator  $\text{to\_combinator } M$  which is extensionally equivalent to  $M$ .

## 8 Program Analysis and Optimisation

The extensional conversion above can be optimised in various ways. In particular, there is no need to convert programs that are already combinators. Also, it is more efficient to convert  $\lambda x.M$  to  $KM$  if  $x$  is not free in  $M$ . Define  $\text{is\_comb}$  by

$$\text{is\_comb} = \text{fix}(\lambda f. \lambda x. F \ x \ K \ (\lambda x_l. \lambda x_r. \text{equal} \ \text{abs\_left} \ x_l \ (KI) \ ((f \ x_l) \ (f \ x_r) \ (KI))))).$$

**Theorem 8.1 (is\_comb\_true)** *For all programs  $M$ , if  $M$  is a combinator then  $\text{is\_comb } M$  reduces to  $K$ .*

**Theorem 8.2 (is\_comb\_false)** *For all programs  $M$ , if  $M$  is not a combinator then  $\text{is\_comb } M$  reduces to  $KI$ .*

The test for deciding if a program  $\lambda x.M$  uses its argument  $x$  can be defined by a term  $\text{binds}$  that detects copies of  $I$  in  $\lambda^*x.M$ . It is given by

$$\text{binds} = \text{fix} \ (\lambda f. \lambda x. \text{equal} \ I \ x \ K \ (F \ x \ (KI) \ (\lambda x_l. \lambda x_r. (f \ x_l) \ K \ (f \ x_r))))).$$

**Theorem 8.3 (binds\_abs\_false)** *For all programs  $\lambda x.M$ , if  $M$  is closed then  $\text{binds} \ (\lambda x.M)$  reduces to  $KI$ .*

**Theorem 8.4 (binds\_abs\_true)** *For all programs  $\lambda x.M$ , if  $x$  is free in  $M$  then  $\text{binds } (\lambda x.M)$  reduces to  $K$ .*

These ideas lead to the definition of the *optimised extensional conversion function* given by

```

to_combinator_opt :=
| O ⇒ O
| λx.M ⇒ (to_combinator_opt (if binds (λx.M) then (λx.M) else (KM)))
| MN ⇒ if is_combinator (MN)
        then (MN)
        else (to_combinator_opt M) (to_combinator_opt N)

```

It is easy to reprise the treatment of `to_combinator` for `to_combinator_opt`, but since these ideas will recur in the next section, there is no particular reason to go through the details here.

## 9 Intensional Conversion to Combinators

Although the conversion functions above preserve extensionality, they lose intensional information, in that an abstraction  $\lambda x.M$  becomes indistinguishable from a star abstraction  $\lambda^*x.M$  or combinator. A conversion function  $f$  *preserves intensions* if it does not lose information, i.e. there is another transformation  $g$  such that  $g(f M)$  reduces to  $M$  for all programs  $M$ .

For example, star abstraction is intensional, since there is an inverse, given by

```

unstar =
| O ⇒ O
| λx.M ⇒ λx.(unstar M)
| KM ⇒ abs_K M
| SMN ⇒ abs_S M N

```

where  $\text{abs\_K} = \lambda x.\lambda y.x$  and  $\text{abs\_S} = \lambda x.\lambda y.\lambda z.x \ z \ (y \ z)$ . The corresponding program, also called `unstar`, is given by program

```

unstar = fix (λf.λx.f x x (λxl.λxr.equal abs_left xl (λz.f (x z))
                                     (equal K xl (abs_K xr) (F xl x (λxll.λxlr.abs_S xlr xr)).

```

**Theorem 9.1 (unstar\_star)** *Star abstraction is intensional, with inverse `unstar`.*

The extensional conversion from programs to combinators can be made intensional, too, by adding tags to record the presence of abstractions and combinators. The *optimised, intensional* conversion of programs to combinators is given by

```

to_combinator_int :=
| O ⇒ O
| λx.M ⇒ abs (to_combinator_int (if binds (λx.M) then (λ*x.M) else (KM)))
| MN ⇒ if is_combinator (MN)
        then com (MN)
        else app (to_combinator_opt M) (to_combinator_opt N)

```

**Theorem 9.2 (to\_combinator\_int\_makes\_combinators)** *If  $M$  is a closed term then `to_combinator_int`  $M$  is a combinator.*

**Theorem 9.3 (to\_combinator\_int\_is\_extensional)** *For all closed terms  $M$  we have `to_combinator_int`  $M \equiv_{\beta\eta SK} M$ .*

The corresponding program `to_comb_int` is given by

```
to_comb_int = fix (λf.λx.F x x (λx_l.λx_r.equal abs_left x_l
  (abs (f(binds x_r x_r (K(x_r K))))))
  (is_comb x (com x) (app (f x_l) (f x_r)))).
```

**Theorem 9.4 (`to_comb_int_to_combinator_int`)** *For all programs  $M$ , there is a reduction  $\text{to\_comb\_int } M \longrightarrow^* \text{to\_combinator\_int } M$ .*

For the conversion in the opposite direction, define `to_program` by

```
to_program :=
| O ⇒ O
| abs M ⇒ unstar(to_program M)
| com M ⇒ M
| app MN ⇒ (to_program M)(to_program N)
```

This can be defined by a term `to_prog`.

**Theorem 9.5 (`to_comb_int_is_intensional`)** *`to_comb_int` is intensional, with inverse given by `to_prog`.*

Summarising, `to_comb_int` maps programs to combinators in a manner that is both extensional and intensional.

## Verification in Coq

The proofs of all the named lemmas and theorems in the paper have been verified using the Coq proof assistant. Details can be found in the source files [6]. This section will reprise some of the key definitions and theorems, to gain some feeling about how well aligned are the manual and automated approaches.

The operators and terms of `lamSF` are given by

```
Inductive operator := | Sop | Fop .
Inductive lamSF : Set :=
| Ref : nat -> lamSF
| Op : operator -> lamSF
| Abs : lamSF -> lamSF
| App : lamSF -> lamSF -> lamSF .
```

The declaration of `operator` declares a type `operator` with two constructors `Sop` and `Fop`. Then the declaration of the type `lamSF` introduces four constructors. `Ref` is used to construct variables, represented by de Bruijn indices of type `nat`, the type of natural numbers. `Op` is used to build the operators  $S$  and  $F$  as `Op Sop` and `Op Fop`. In most situations, all operators are treated uniformly, which is exploited by giving them a separate type. `Abs` constructs abstractions and `App` constructs applications. In this manner, the function  $\lambda x.\lambda y.xy.SF$  is represented by

`Abs(Abs(App(App(App(Ref 1)(Ref 0))(Op Sop))(Op Fop)))` .

The biggest gap between this representation and the paper representation is the use of de Bruijn indices for variables. For example, the requirement `maxvar M = 1` means that `M` has exactly one free variable (indexed by 0).

The Coq versions of the named results in the paper are given in Figure 9. Most of the unexplained notation, such as `confluence` should be self-explanatory. Note,

```

Theorem confluence_lamSF_red: confluence lamSF lamSF_red.
Theorem irreducible_iff_normal:
  forall M, irreducible M lamSF_red1 <=> normal M.
Theorem programs_are_factorable : forall M, program M -> factorable M.
Theorem equal_programs : forall M, program M -> lamSF_red (App (App equal M) M) k_op.
Theorem unequal_programs :
  forall M N, program M -> program N -> M<>N ->
    lamSF_red (App (App equal M) N) (App k_op i_op).
Lemma star_equiv_abs : forall M, beta_eta_eq (star M) (Abs M) .
Theorem no_homomorphism: forall h, homomorphism h -> False.
Theorem to_combinator_makes_combinators :
  forall M, closed M -> combinator (to_combinator M).
Theorem to_combinator_is_extensional : forall M, beta_eta_eq M (to_combinator M).
Theorem to_combinator_to_comb:
  forall M, program M -> lamSF_red (App to_comb M) (to_combinator M).
Theorem is_comb_true: forall M, program M -> combinator M -> lamSF_red (App is_comb M) k_op.
Theorem is_comb_false:
  forall M, program M -> (combinator M -> False) ->
    lamSF_red (App is_comb M) (App k_op i_op).
Theorem binds_abs_false :
  forall M, program (Abs M) -> closed M ->
    lamSF_red (App binds (Abs M)) (App k_op i_op).
Theorem binds_abs_true :
  forall M, program (Abs M) -> maxvar M = 1 ->
    lamSF_red (App binds (Abs M)) k_op.
Theorem unstar_star : forall M, normal M -> lamSF_red (App unstar (star M)) (Abs M).
Lemma wait_ext : forall M N, beta_eta_eq (wait M N) (App M N).
Lemma tag_ext : forall T M, beta_eta_eq (tag T M) M.
Lemma eager_is_eager : forall M N, factorable N -> lamSF_red (eager M N) (App M N).
Theorem to_combinator_int_makes_combinators :
  forall M, closed M -> combinator (to_combinator_int M).
Theorem to_combinator_int_is_extensional :
  forall M, closed M -> beta_eta_eq M (to_combinator_int M).
Theorem to_comb_int_to_combinator_int:
  forall M, program M ->
    lamSF_red (App to_comb_int M) (to_combinator_int M).
Theorem to_comb_int_is_intensional :
  forall M, program M -> lamSF_red (App to_prog (App to_comb_int M)) M.

```

Fig. 3. Theorems Verified in Coq

however, that `homomorphism` is here defined to be a homomorphism from `lamSF` to `lambda` rather than a homomorphism in general. Also, `beta_eta_eq` is here the equivalence relation generated from  $\beta\eta SK$ -reduction, and not just from  $\beta$ - and  $\eta$ -reduction.

## 10 Fresh Approaches

Having established the basic machinery of  $\lambda SF$ -calculus and seen something of its expressive power, it is interesting to consider, at least in outline, how it suggests fresh approaches to some issues.

**Gödelisation** Although  $\lambda$ -calculus is Turing-complete, in the sense of being able to compute any number that a Turing machine can, there are strong limits to its ability to compute functions of  $\lambda$ -terms. For example, equality of closed normal  $\lambda$ -abstractions is not definable as  $\lambda$ -abstraction [1]. Nor is it possible to so define the Gödel number of a closed normal form. With a little effort, the conversion of programs to combinators can be extended to support Gödelisation.

**Self-interpretation** Self-interpretation is used to support programming language implementation within the language itself. In particular, it can be used to impose an evaluation strategy upon a confluent calculus such as  $\lambda$ -calculus [14] or  $SF$ -calculus [9]. Traditionally, the first step in self-interpretation is to use meta-level calculations to *quote* a program, to produce a data structure that is suitable for analysis. Since programs in  $\lambda SF$ -calculus are already data structures, there is no need for quotation. Indeed, evaluation strategies can be defined within the calculus,

without the need for any meta-level analysis.

**Term constructors** In the traditional  $\lambda$ -calculus account, the same  $\lambda$ -abstraction may have several different meanings. For example, the natural number zero may be represented as  $\lambda f.\lambda x.x$ , in which  $f$  is applied zero times to  $x$ . Also, the boolean for falsehood may be represented by  $\lambda x.\lambda y.y$  in which the second branch, represented by its second argument, is taken. However,  $\lambda f.\lambda x.x$  and  $\lambda x.\lambda y.y$  are equivalent under renaming of bound variables, so that the same term has two different meanings. Traditionally, these have been distinguished by either introducing constructors, such as **Zero** and **False**, or adding types, such as **Nat** and **Bool**, or both. Now, we can tag these abstractions with information about their status as constructors, or their types. Similarly, constructor *arities* can be recorded by using **wait**.

**Pattern calculus** In  $\lambda SF$ -calculus, it should be possible to give a complete account of constructor equality and pattern-matching by manipulating intensional information.

**Type checking** Similarly, once terms are tagged with type information, the calculus should support type checking and type inference.

**Evaluation strategy** Confluent rewriting systems support a natural model of program optimisation by changing the order in which sub-expression are evaluated. However, sequential execution requires that an evaluation strategy be imposed. As with intensional information, different strategies give rise to a variety of different calculi [16]. These can be captured by using terms such as **wait** and **eager** to control evaluation order.

**Partial evaluation** Once programs are represented by normal forms, it is much easier to understand the nature of partial evaluation, of static arguments versus dynamic arguments, etc [11]. As before, these analyses should now be representable as programs.

**Domain specific languages** Users are driven to create their own, domain-specific programming languages because general purpose languages prove to be sub-optimal for their needs. One approach is to grow a language from a small core [19,10]. This will be easier once program analysis and evaluation strategies are definable.

## 11 Conclusions

$\lambda SF$ -calculus combines the best features of  $\lambda$ -calculus and combinatory calculi within a single calculus in that  $\lambda$ -abstraction provides a natural account of functionality through its  $\beta$ -reduction, while combinators provide a natural account of data structures, once the factorisation operator  $F$  is supported. Together, they show how programs and data structures can both be identified with the closed normal forms of  $\lambda SF$ -calculus, so that they may be applied or analysed at any time. Further, the combinators can be used to tag programs with additional, intensional information, e.g. about constructors or types, or to control evaluation strategy by making applications wait before reducing.

The identification of programs and data structures also removes a layer of indirection from program analysis. There is no need to quote or Gödelise abstractions. Nor is there need for a separate state machine, to evaluate programs expressed on a tape. The ramifications may extend to all aspects of programming language design



and implementation, including analysis and optimisation.

Like pattern calculus and  $SF$ -calculus,  $\lambda SF$ -calculus supports powerful collection of generic queries for searching and updating data structures. However, the earlier calculi were far removed from current experience, making adoption difficult. By contrast,  $\lambda SF$ -calculus merely adds a couple of operators to the popular  $\lambda$ -calculus approach, which makes migration much easier.

In conclusion,  $\lambda SF$ -calculus adds intensionality to the extensional nature of  $\lambda$ -calculus, so that one can query the internal structure of arbitrary closed normal forms, and treat programs as data structures.

## References

- [1] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North Holland, 1984. revised edition.
- [2] Henk Barendregt. Self-interpretations in lambda calculus. *J. Funct. Program*, 1(2):229–233, 1991.
- [3] Michel Bel. A recursion theoretic self interpreter for the lambda-calculus. <http://www.belxs.com/michel/#selfint>.
- [4] Alessandro Berarducci and Corrado Böhm. A self-interpreter of lambda calculus having a normal form. In *CSL*, pages 85–99, 1992.
- [5] Barry Jay. *Pattern Calculus: Computing with Functions and Structures*. Springer, 2009.
- [6] Barry Jay. LamSF repository of proofs in Coq. <https://github.com/Barry-Jay/lambdaSF>, February 2016.
- [7] Barry Jay and Thomas Given-Wilson. A combinatory account of internal structure. *Journal of Symbolic Logic*, 76(3):807–826, 2011.
- [8] Barry Jay and Delia Kesner. First-class patterns. *Journal of Functional Programming*, 19(2):191–225, 2009.
- [9] Barry Jay and Jens Palsberg. Typed self-interpretation by pattern matching. In *Proceedings of the 2011 ACM Sigplan International Conference on Functional Programming*, pages 247–58, 2011.
- [10] Barry Jay and Jose Vergara. Growing a language in pattern calculus. In *Theoretical Aspects of Software Engineering (TASE), 2013 International Symposium on*, pages 233–240. IEEE, 2013.
- [11] N.D. Jones, C.K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. International Series in Computer Science. Prentice Hall International, 1993.
- [12] Stephen C. Kleene.  $\lambda$ -definability and recursiveness. *Duke Math. J.*, pages 340–353, 1936.
- [13] J.W. Klop. *Combinatory Reduction Systems*. PhD thesis, Mathematical Center Amsterdam, 1980. Tracts 129.
- [14] Torben Æ. Mogensen. Efficient self-interpretations in lambda calculus. *Journal of Functional Programming*, 2(3):345–363, 1992. See also DIKU Report D-128, Sep 2, 1994.
- [15] Torben Æ. Mogensen. Linear-time self-interpretation of the pure lambda calculus. *Higher-Order and Symbolic Computation*, 13(3):217–237, 2000.
- [16] G.D. Plotkin. Call-by-name, call-by-value and the  $\lambda$ -calculus. *Theoretical Computer Science*, 1, 1975.
- [17] John C. Reynolds. Definitional interpreters for higher-order programming languages. In *Proceedings of 25th ACM National Conference*, pages 717–740. ACM Press, 1972. The paper later appeared in *Higher-Order and Symbolic Computation*.
- [18] Fangmin Song, Yongsun Xu, and Yuechen Qian. The self-reduction in lambda calculus. *Theoretical Computer Science*, 235(1):171–181, March 2000.
- [19] Guy L. Steele. Growing a language. *Higher-Order and Symbolic Computation*, 12(3), 1999.
- [20] Terese. *Term Rewriting Systems*, volume 53 of *Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

# A Monad for Randomized Algorithms

Tyler Barker<sup>1</sup>

*Mathematics  
Tulane University  
New Orleans, LA, USA*

---

## Abstract

In this paper, we introduce a monad of random choice for domains that does not suffer from the main two drawbacks of the probabilistic powerdomain. It is not known whether any Cartesian closed category of domains is closed under the probabilistic powerdomain, but the Cartesian closed category BCD is closed under this monad of random choice. Also, there is no distributive law between the probabilistic powerdomain and any of the nondeterministic powerdomains, but there is a distributive law between the monad of random choice and the lower powerdomain. In order to work with the convex powerdomain, an alteration to the monad of random choice is made, so that the Cartesian closed categories **RB** and **FS** are closed under this construction. Then, in these categories, there is a distributive law between this monad and the convex powerdomain. This work is based on the uniform continuous random variables of Goubault-Larrecq and Varacca, which do not form a monad. This paper gives motivation for this model and changes the definition of the Kleisli extension of Goubault-Larrecq and Varacca so that it is monotone, which was the problem with their definition.

*Keywords:* Probabilistic powerdomain, Cartesian closed category, random variable, distributive law

---

## 1 Introduction

Starting with Dana Scott's model of the untyped lambda calculus, domain theory has been largely successful in providing models of computation. The use of domain theory has expanded to provide denotational semantics for many computational effects, such as continuations and nondeterminism, using Moggi's [16] monadic approach. One type of computation that has been problematic to model, however, is probabilistic computation. The most well known monad of probabilistic computation is the probabilistic powerdomain, first defined by Saheb-Djahromi in 1980 [18]. However, this monad has two major flaws [11]. First, there is no distributive law between the probabilistic powerdomain and any of the three nondeterministic powerdomains [24]. According to Beck's Theorem [4], the composition of two monads is a monad if and only if the monads satisfy a distributive law. Thus, to generate a monad from the probabilistic powerdomain and any of the monads for nondeterministic choice, new laws must be added, an approach explored independently by

---

<sup>1</sup> Email: [tbarker@tulane.edu](mailto:tbarker@tulane.edu)

Tix [22, 23] and Mislove [12]. Second, it is not known whether any Cartesian closed category of domains is closed under the probabilistic powerdomain. The category of coherent domains is closed under this construction, but it is not Cartesian closed.

To address these flaws, work has been done to develop alternate models of probabilistic computation. Varacca and Winskel [24, 25] constructed what they called *indexed valuation monads*. These monads weaken the laws of probabilistic choice, no longer requiring that  $p +_r p = p$ , where  $p +_r q$  denotes choosing  $p$  with probability  $r$  and  $q$  with probability  $1 - r$ . In this setting, it is possible to satisfy a distributive law with the nondeterministic powerdomains.

Mislove [13] built upon this work, using an indexed valuation model to define a monad of finite random variables. The Cartesian closed categories RB and FS were shown to be closed under this construction. Later, Goubault-Larrecq and Varacca [9] proposed a model of continuous random variables over the Cartesian closed category BCD, but the model did not form a monad in this category [14]. The model that this paper describes is based upon these continuous random variables, in particular, the uniform continuous random variables. In this construction, computation is allowed to take different branches based on the flips of an unbiased coin.

The contribution of this paper is to redefine the Kleisli extension mapping proposed in Goubault-Larrecq and Varacca's paper so that the resulting construction forms a monad on the category BCD. We also obtain a distributive law with this monad and the lower powerdomain. Another slight alteration can be made to the model in order to work in the Cartesian closed categories RB and FS, since the convex powerdomain does not necessarily stay in BCD. Then, working in RB and FS, this altered monad is shown to satisfy a distributive law with respect to the convex powerdomain.

The monad laws and the distributive law are both defined by a few commutative diagrams. Verifying that these diagrams do indeed commute is usually straightforward, but it can be very tedious. These details are omitted from this paper for readability and space concerns, but they will be included in the author's upcoming thesis.

## 2 Background

### 2.1 Domain Theory

It will be assumed that the reader has some familiarity with domain theory. For more information, consult [1, 7].

A *poset* is a partially ordered set. A subset of a poset is an *antichain* if no two distinct elements of the poset are comparable.

A nonempty subset is *directed* if each pair of its elements has an upper bound also within the subset. A poset is *directed complete* if each of its directed subsets has a least upper bound. A *dcpo* is a directed complete partial order. Maps between dcpos that are monotone and preserve suprema of directed sets are called *Scott continuous*.

The following is the least fixed-point theorem for Scott continuous functions:

**Theorem 2.1** *Let  $D$  be a dcpo with a least element  $\perp$ . Then every Scott continuous self-map  $f : D \rightarrow D$  has a least fixed-point. It is given by  $\cup_{n \in \mathbb{N}} f^n(\perp)$ .*

Now we define the Scott and Lawson topologies.

**Definition 2.2** Let  $D$  be a dcpo. A subset  $U$  is *Scott closed* if it is a lower set and is closed under directed suprema of subsets.

The Scott closed sets are closed under all intersections and finite unions, so they are the closed sets of a topology, which is called the Scott topology. The Scott continuous functions defined above are precisely the functions that are continuous with respect to this topology.

**Definition 2.3** Let  $D$  be a dcpo. The *Lawson topology* on  $D$  is the smallest topology containing the Scott open sets and all sets of the form  $D \setminus \uparrow x$ .

If  $D$  is a dcpo and  $x, y \in D$ , then  $x$  *approximates*  $y$  (denoted  $x \ll y$ ) iff for every directed set  $S$  with  $y \leq \sup S$ , there is some  $s \in S$  such that  $x \leq s$ . Let  $\downarrow y = \{x \in D \mid x \ll y\}$ . A dcpo  $D$  is a *domain* iff  $\forall d \in D, \downarrow d$  is directed and  $\sup \downarrow d = d$ . Note that the Lawson topology on a domain is Hausdorff.

**Definition 2.4** A domain  $D$  is *coherent* if it is compact in the Lawson topology.

## 2.2 Category Theory

In this paper, we work within Cartesian closed categories as these are necessary to model lambda calculi [2].

**Definition 2.5** A category is a *Cartesian closed category (CCC)* if it has a terminal object, products, and exponentials.

We are interested in three particular Cartesian closed categories of domains: BCD, RB, and FS. The maximal Cartesian closed categories of domains were characterized by Jung [10]. Here are descriptions of the Cartesian closed categories of domains we will need. Note that in each case, the morphisms in the category are the Scott-continuous maps.

**Definition 2.6** A domain is *bounded complete* if every subset with an upper bound has a least upper bound. Equivalently, a domain is *bounded complete* if every nonempty subset has a greatest lower bound. BCD denotes the category of bounded complete domains and Scott continuous maps.

**Definition 2.7** A self-map on a domain  $D$  is a *deflation* if it is less than the identity map in the pointwise order and has a finite image.

**Definition 2.8** A domain is a *retract of a bifinite domain*, or an *RB-domain*, if there exists a directed family  $(f_i)_{i \in I}$  of Scott continuous deflations whose supremum is the identity map. RB denotes the category of RB-domains and Scott continuous maps.

**Definition 2.9** A self-map on a domain  $D$  is *finitely separated* from the identity map if there exists a finite set  $M \subseteq D$  such that  $\forall x \in D, \exists m \in M. f(x) \leq m \leq x$ .

**Definition 2.10** A domain is a *finitely separated domain*, or an *FS-domain*, if there exists a directed family  $(f_i)_{i \in I}$  of Scott continuous self-maps, each finitely separated from the identity map, whose supremum is the identity map. **FS** denotes the category of FS-domains and Scott continuous maps.

BCD is a subcategory of RB, which is a subcategory of FS. FS is a maximal Cartesian closed category; however, it is a frustrating open question whether RB is a proper subcategory of FS.

Finally, all three of these categories are subcategories of COH, the category of coherent domains and Scott continuous maps, but COH is not Cartesian closed.

### 2.3 Monads

A monad is a construction from category theory that has proven to be very useful in modeling computational effects.

**Definition 2.11** A *monad* on a category  $C$  is a triple,  $(T, \eta, \mu)$ , where  $T$  is an endofunctor and  $\eta : \text{Id}_C \rightarrow T$ ,  $\mu : T^2 \rightarrow T$  are natural transformations such that the following diagrams commute:

$$\begin{array}{ccc} TX & \xrightarrow{\eta_{TX}} & T^2X \\ T\eta_X \downarrow & \searrow \text{id}_{TX} & \downarrow \mu_X \\ T^2X & \xrightarrow{\mu_X} & TX \end{array} \qquad \begin{array}{ccc} T^3X & \xrightarrow{\mu_{TX}} & T^2X \\ T\mu_X \downarrow & & \downarrow \mu_X \\ T^2X & \xrightarrow{\mu_X} & TX \end{array}$$

The natural transformation  $\eta$  is called the unit of the monad, and  $\mu$  is the multiplication.

There is an alternate characterization of a monad that uses a Kleisli extension in place of the multiplication. An endofunctor  $T$  is a monad if, for any map  $f : X \rightarrow TY$ , there is a Kleisli extension  $f^\dagger : TX \rightarrow TY$ , and the following laws hold:

- (i)  $\eta^\dagger = \text{id}$
- (ii)  $h^\dagger \circ \eta_D = h$
- (iii)  $k^\dagger \circ h^\dagger = (k^\dagger \circ h)^\dagger$

Given the Kleisli extension, the multiplication is defined by  $\mu = \text{id}_{TX}^\dagger$ . Conversely, given the multiplication and a function  $f : X \rightarrow TY$ , then  $f^\dagger = \mu \circ T(f)$ .

### 2.4 Powerdomains

Nondeterminism is modeled in domain theory by powerdomains which are built by considering nondeterministic choice as an idempotent, commutative, and associative operation [15]. This is equivalent to the algebraic definition of a semilattice, so the powerdomains are simply free ordered semilattice domains over a domain.

Starting with a poset having a commutative, idempotent operation  $+$ , assuming  $x \leq x + y$  results in a sup-semilattice, assuming  $x \geq x + y$  results in a inf-semilattice, and an ordered semilattice will result from not assuming any relation between  $x$  and  $x + y$ . The lower, or Hoare, powerdomain is the free sup-semilattice over a domain, the upper, or Smyth, powerdomain is the free inf-semilattice over a domain, and

the convex, or Plotkin, powerdomain is the free ordered semilattice over a domain. These powerdomains have nice topological characterizations which will be used in this paper.

**Definition 2.12** For a domain  $D$ , the *lower powerdomain* is  $\Gamma_0(D)$ , the family of nonempty Scott closed subsets of  $D$ , ordered by inclusion.

**Definition 2.13** A subset  $S$  of a topological space  $X$  is *saturated* if it is the intersection of the open sets that contain it.

For a poset with the Scott topology, saturated sets are simply the upper sets.

**Definition 2.14** For a domain  $D$ , the *upper powerdomain* is  $SC(D)$ , the family of nonempty, saturated, and Scott compact subsets of  $D$ , ordered by reverse inclusion.

**Definition 2.15** For a domain  $D$ , a subset  $L \subseteq D$  is a *lens* if  $L$  is Scott compact and  $L = \overline{L} \cap \uparrow L$ , where  $\overline{L}$  is the Scott closure of  $L$ .

**Definition 2.16** The *Egli-Milner order* is defined by:

$$A \sqsubseteq_{EM} B \Leftrightarrow A \subseteq \downarrow B \wedge B \subseteq \uparrow A$$

**Definition 2.17** For a coherent domain  $D$ , the *convex powerdomain* is  $Lens(D)$ , the family of nonempty lenses, with the Egli-Milner order.

All three of the above categories, BCD, RB, and FS, are closed under the lower and upper powerdomains, but only RB and FS are closed under the convex powerdomain.

## 2.5 Randomized Computation

We consider a randomized computation to be any program or algorithm that uses some source of randomness to guide its computation. This includes assigning a random value to a variable or using a conditional expression that branches based on the output of a random process. If the probability distribution of the random source is known, we can view this as probabilistic computation. Abstractly, probabilistic computation is usually represented as a probabilistic choice operator,  $p +_r q$ , where  $p$  is chosen with probability  $r$  and  $q$  is chosen with probability  $1 - r$ .

Probabilistic Turing machines were first defined by de Leeuw *et al* in 1956 [5]. These machines are the same as normal Turing machines with an attached random device. This device prints 0's and 1's to a tape, with 1's occurring with probability  $p$  and 0's occurring with probability  $1 - p$ , where  $0 < p < 1$ . This tape can then be used as an input tape for the Turing machine. It was shown that as long as  $p$  is computable, then these machines cannot compute anything that a deterministic machine cannot compute. However, it may be possible that a probabilistic machine can compute something faster than any deterministic machine could [8].

Randomized computation first gained prominence when Rabin [6] introduced a randomized algorithm for finding the nearest pair in a set of  $n$  points. This algorithm had a linear average runtime, faster than the  $n \log n$  runtime of the fastest known deterministic algorithm. More well known are the algorithms of Solovay and Strassen [21] and Rabin [17] for determining if a number is prime. These algorithms

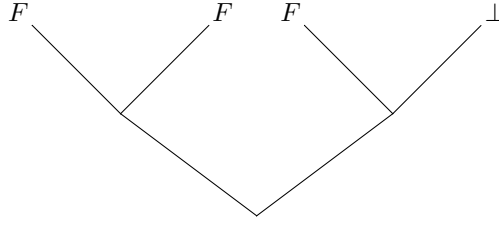


Fig. 1. One possible iteration of a simplified Miller-Rabin test on a composite number.

run in polynomial time (with a small error probability), and they were discovered over 20 years before the AKS primality test [3], the first known deterministic algorithm for recognizing prime numbers in polynomial time.

### 3 The Functor

This work is inspired by a model of uniform continuous random variables first proposed by Goubault-Larrecq and Varacca [9]. In their paper, it was shown that the category of bounded complete domains (BCD) is closed under a similar construction. However, their assertion that the construction forms a monad in BCD was incorrect, since the proposed Kleisli extension failed to be monotone, thus not Scott continuous.

The basic idea of a random variable model is to separate the random choices from the domain itself. In the probabilistic powerdomain, the probability distributions are placed on the underlying domain. In a model of random variables, random bits are generated by coin flips, and then a random variable is defined from these random outcomes to the underlying domain. In the probabilistic powerdomain, for an element  $d$ , making a choice between  $d$  and  $d$  is the same as just  $d$ , since the probabilities are the same. In the model described here, there is a distinction between choosing  $d$  or  $d$  and  $d$  itself, even though the probabilities are the same. In the first case, a random bit is still chosen, so programmatically, this is distinct from the latter case where no such choice is made.

#### 3.1 Motivation for the Functor

One of most well known randomized algorithms is the Miller-Rabin primality test. To test whether a given number  $n$  is prime, a random number is chosen between 2 and  $n - 2$ . Tests using modular arithmetic are performed with this random number before determining whether the given number is composite or probably prime. The test can be run in polynomial time, but has a possible one-sided error, putting primality testing in the complexity class of randomized polynomial time (RP). A test on a prime number will always return “probably prime”, but sometimes, a test on a composite number will also return “probably prime”. Thus, if the test returns “composite”, there is no chance for error, but a return of “probably prime” always has a chance of error. For a composite number, at most  $\frac{1}{4}$  of the possible random choices between 2 and  $n - 2$  will result in the test returning “probably prime”. To minimize the error probability, we can repeat the test (choosing a new random number) only when the test returns “probably prime”. Running the test  $m$  times

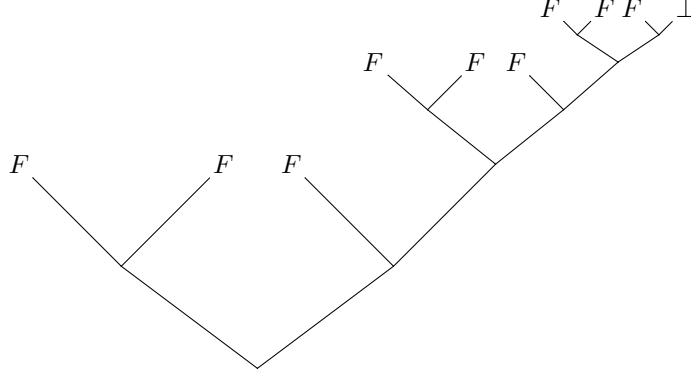


Fig. 2. Three iterations of a hypothetical Miller-Rabin test.

results in an error probability of at most  $\frac{1}{4^m}$ .

Figure 1 shows the possible outcomes of a hypothetical Miller-Rabin test on a composite number. For simplicity, it is assumed that a random number between 2 and  $n - 2$  can be properly chosen using just two coin flips. Each coin flip is represented by a branching of the binary tree. The top of the tree is labeled with the return values of the test using the random numbers chosen by the resulting outcome of two coin flips. If the test returns “composite”, an “F” is used whereas “ $\perp$ ” denotes “probably prime”. A “T” is not used since a Miller-Rabin test never confirms that a number is prime. If we wish to minimize the error probability, we can choose to run the test again, which will expand the tree wherever a “ $\perp$ ” is found.

Figure 2 shows the possible outcomes of using Miller-Rabin a maximum of three times on the same composite number. This can be extended similarly to an infinite tree with a zero probability of error.

### 3.2 The Functor Definition

Let  $\{0, 1\}^\infty = \{0, 1\}^* \cup \{0, 1\}^\omega$  be the set of finite and infinite words of alphabet  $\{0, 1\}$ , with the prefix order ( $w \leq w'$  if  $w$  is a prefix of  $w'$ ). The symbol  $*$  is used to denote the concatenation operation. In this setting, a 0 represents getting tails on a coin flip, and a 1 signifies heads. If a fair coin is used, then for any word, the probability associated with the word is  $\frac{1}{2^n}$ , where  $n$  is the length of the word. For example, the probability of 10, which represents getting heads and then tails, is  $\frac{1}{4}$ .

**Definition 3.1** An *antichain* of  $\{0, 1\}^\infty$  is a subset of words such that no two distinct words are comparable (no word is a prefix of another word). An antichain  $M$  is *full* if  $\forall w \in \{0, 1\}^\omega, \exists z \in M, z \leq w$ . Put another way,  $\{0, 1\}^\omega \subseteq \uparrow M$ , or  $M \sqsubseteq_{EM} \{0, 1\}^\omega$ . Denote the nonempty, full antichains by  $FAC(\{0, 1\}^\infty)$ .

Using coin flips in a program results in a branching of computation that can be represented as a binary tree. The final possible outcomes will be located at the leaves of this tree, which must form an antichain. This antichain is required to be full since for any coin flip, it is possible to get either heads or tails, and both outcomes must be accounted for.

**Definition 3.2** For a category of domains, the random choice functor,  $RC$ , is de-



fined on objects by

$$RC(D) = \{(M, f) \mid M \in FAC(\{0, 1\}^\infty, f : M \rightarrow D)\}$$

where  $f$  is Scott continuous (giving  $M$  the subspace topology from the Scott topology of  $\{0, 1\}^\infty$ ). For a morphism,  $a : D \rightarrow D'$ , and  $(M, f) \in RC(D)$ , we define

$$RC(a)(M, f) = (M, a \circ f)$$

We order  $RC(D)$  by  $(M, f) \sqsubseteq (N, g)$  iff  $M \sqsubseteq_{EM} N$  and  $w \leq z \Rightarrow f(w) \sqsubseteq g(z), \forall w \in M, z \in N$ . Since the antichains are required to be full,  $M \sqsubseteq_{EM} N$  is equivalent to  $M \subseteq \downarrow N$ , or dually,  $N \subseteq \uparrow M$ . Another characterization for the order on functions is  $\forall z \in N, f \circ \pi_M(z) \sqsubseteq g(z)$ , where  $\pi_M(z)$  sends  $z$  to the unique element of  $M$  below  $z$ .

**Theorem 3.3** *If  $D$  is a bounded complete domain, then so is  $RC(D)$ .*

## 4 The RC Monad

To show that the functor  $RC$  forms a monad, the unit and Kleisli extension (or multiplication) of the monad must be exhibited. For a domain  $D$  and  $d \in D$ , the unit,  $\eta : D \rightarrow RC(D)$  is defined by

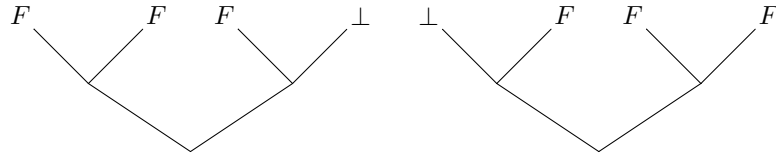
$$\eta(d) = (\epsilon, \chi_d)$$

where  $\epsilon$  is the antichain only containing the empty word, and  $\chi_d$  is the constant function whose value is  $d$ .

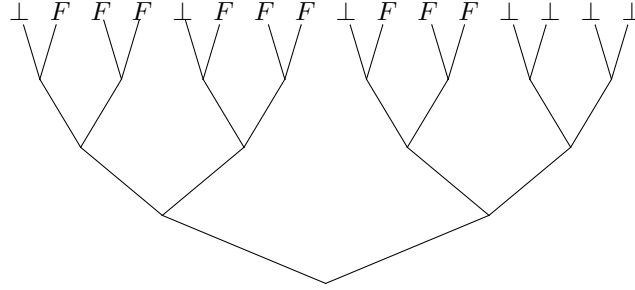
### 4.1 Motivation for the Kleisli extension

The Kleisli extension of a monad  $T$  can be hard to think about intuitively since we normally do not work with functions from  $D$  to  $T(E)$ . However, the Kleisli extension is important in lifting binary operations on the underlying structures to binary operations on the monadic structures. If we have a binary operation  $* : D \times E \rightarrow F$ , the Kleisli extension lifts this operation to  $*^\dagger : T(D) \times T(E) \rightarrow T(F)$ . This is achieved by setting  $*^\dagger = (\lambda a. T(\lambda b. a * b))^\dagger$ .

Suppose that we have a Miller-Rabin test performed on two composite numbers with the following possible outcomes:



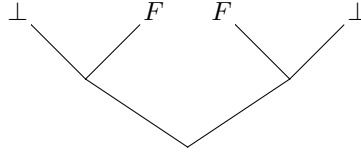
How should the binary operation **or** be lifted? It may seem natural to perform the two tests one after the other, resulting in:



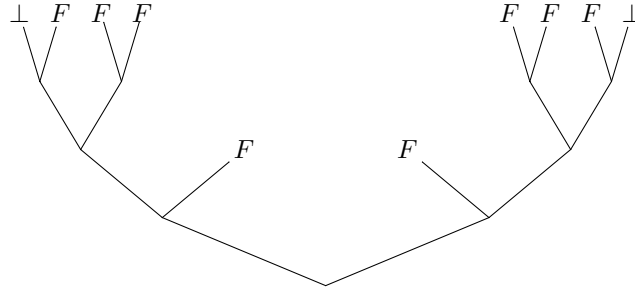
The probability of error in this case is  $\frac{7}{16}$ , assuming we use a fair coin. However, this method has two main flaws.

- (i) How do we handle the infinite case? If the first random test can use infinitely many coin flips, then the second test will never even start.
- (ii) The Kleisli extension that results in this behavior is not monotone. Therefore, it does not form a monad in a category we want.

Instead, consider feeding the result of each coin flip to both tests concurrently. For two coin flips, our example would look like:



To properly compare it with the sequential case, we should use the same maximum number of coin flips. Feeding all four coin flips to both Miller-Rabin tests results in:



which only has an error probability of  $\frac{1}{8}$ . If the error possibility for each number had coincided, then the error probability would have been smaller,  $\frac{1}{16}$ .

It some cases, it may be desirable to have two random processes run sequentially instead of the concurrent behavior described here. However, for a randomized algorithm like Miller-Rabin, which has a fixed desired output, this is unnecessary. These algorithms are represented by possibly infinite trees that have a zero error probability. Combining these trees as described above results in another tree with a zero error probability. In fact, the error probability can decrease more quickly using this method. But if it is necessary to have a sequential composition of random processes, then we must move outside of the  $RC$  monad to accomplish this. We can create a function on  $RC(D) \times \mathbb{N}$  that takes a random process and uses  $n$  coin

flips to output an element of  $D$ . This function is not deterministic, so the output would need to be in  $T(D)$  for some other monad  $T$ . For example, in Haskell, we can use the standard random library to simulate coin flips, and  $T$  in this case would be Haskell's IO monad. Composing two such actions would result in the random choices occurring sequentially. More generally, we can use a probabilistic monad such as the indexed valuations or finite random variables as the codomain of this function.

#### 4.2 Kleisli Extension of the Monad

Consider  $h : D \rightarrow RC(E)$ . For an element  $(M, f) \in RC(D)$ , each  $w$  represents one possible outcome of coin flips. For each  $w$ ,  $h \circ f(w)$  gives another randomized algorithm, in  $RC(E)$ . Thus, there is a random choice of random algorithms, and the Kleisli extension has to convert this into one randomized algorithm in  $(N, g) \in RC(E)$ . Instead of using all of  $h \circ f(w)$ , for each  $w$ , we use the coin flips represented by  $w$  and feed them into  $h \circ f(w)$ . If the first coin flip was “heads” moving towards  $w$ , then we assume that the first coin flip will be “heads” when running  $h \circ f(w)$ . Thus, our extension only considers the part of  $\pi_1 \circ h \circ f(w)$  that is on the same “branch” of the tree as  $w$ , namely  $\uparrow w \cup \downarrow w$ .

**Definition 4.1** For  $(M, f) \in RC(D)$ , the first component of the Kleisli extension,  $\pi_1 \circ h^\dagger(M, f)$  will give an antichain that is bigger than the original  $M$ . It is defined as follows:

$$\pi_1 \circ h^\dagger(M, f) = \bigcup_{w \in M} \text{Min}(\uparrow w \cap \uparrow \pi_1 \circ h \circ f(w))$$

where  $\text{Min}(W)$  gives the minimal words of  $W$ . The second component of the Kleisli extension gives a function from the first component into  $E$ . For a given  $z$  in the first component, this is defined by:

$$((\pi_2 \circ h^\dagger)(M, f))(z) = g(\pi_N(z)) \text{ where } (N, g) = h \circ f \circ \pi_M(z)$$

Since the first component given by the Kleisli extension is bigger than  $M$ , the function  $f$  may not be defined on  $z$ . Therefore,  $\pi_M(z)$  is used followed by  $h \circ f$  to pick a randomized algorithm  $(N, g) \in RC(E)$ . Similarly,  $g$  may not be defined on  $z$ , but there is a unique element of  $N$ ,  $\pi_N(z)$ , where it is defined.

**Proposition 4.2**  $h^\dagger$  is monotone.

**Proof.**  $(M, f) \leq (N, g)$  means that  $N \subseteq \uparrow M$  (thus,  $\uparrow N \subseteq \uparrow M$ ) and  $w \leq z \Rightarrow f(w) \leq g(z)$  for any  $w \in M, z \in N$ .

$$\begin{aligned} \uparrow \pi_1 \circ h^\dagger(M, f) &= \uparrow \bigcup_{w \in M} \text{Min}(\uparrow w \cap \uparrow \pi_1 \circ h \circ f(w)) \\ &= \bigcup_{w \in M} \uparrow \text{Min}(\uparrow w \cap \uparrow \pi_1 \circ h \circ f(w)) \\ &= \bigcup_{w \in M} (\uparrow w \cap \uparrow \pi_1 \circ h \circ f(w)) \end{aligned}$$

The same applies to  $(N, g)$ , so we just need to show that

$$\bigcup_{z \in N} (\uparrow z \cap \uparrow \pi_1 \circ h \circ g(z)) \subseteq \bigcup_{w \in M} (\uparrow w \cap \uparrow \pi_1 \circ h \circ f(w))$$

For each  $z \in N$ , there is a  $w \in M$  that is below  $z$ . In this case,  $\uparrow z \subseteq \uparrow w$  and  $\uparrow(\pi_1 \circ h \circ g(z)) \subseteq \uparrow(\pi_1 \circ h \circ f(w))$  since  $g(z) \geq f(w)$  and  $h$  is monotone. Thus,  $(\uparrow z \cap \uparrow(\pi_1 \circ h \circ g(z))) \subseteq (\uparrow w \cap \uparrow(\pi_1 \circ h \circ f(w)))$ .

Now we check the functions. For  $w \in (\pi_1 \circ h^\dagger(M, f))$  and  $z \in (\pi_1 \circ h^\dagger(N, g))$  with  $w \leq z$ , we must show that  $(\pi_2 \circ h^\dagger(M, f))(w) \leq (\pi_2 \circ h^\dagger(N, g))(z)$ .

Let  $\pi_M(w)$  equal the unique word in  $M$  below  $w$ . Since  $w \leq z$ ,  $\pi_M(w) \leq \pi_N(z)$ , and  $f \circ \pi_M(w) \leq g \circ \pi_N(z)$ . Since  $h$  is monotone,  $h \circ f \circ \pi_M(w) \leq h \circ g \circ \pi_N(z)$ .

Again, since  $w \leq z$ ,  $\pi_{\pi_1 \circ h \circ f \circ \pi_M}(w) \leq \pi_{\pi_1 \circ h \circ g \circ \pi_N}(z)$ , and we have

$$\begin{aligned} (\pi_2 \circ h^\dagger(M, f))(w) &= (\pi_2 \circ h \circ f \circ \pi_M(w))(\pi_{\pi_1 \circ h \circ f \circ \pi_M}(w)(w)) \\ &\leq (\pi_2 \circ h \circ g \circ \pi_N(z))(\pi_{\pi_1 \circ h \circ g \circ \pi_N}(z)(z)) \\ &= (\pi_2 \circ h^\dagger(N, g))(z) \end{aligned} \quad \square$$

**Proposition 4.3**  $h^\dagger$  is Scott continuous.

**Theorem 4.4** The functor  $RC$  forms a monad in the category  $BCD$ .

## 5 Distributive Laws and Extending the Monad

One of the downsides with the probabilistic powerdomain is that it does not satisfy a distributive law with any of the nondeterministic powerdomains. However, we can show that our monad  $RC$  does satisfy a distributive law with respect to the lower powerdomain in the category  $BCD$ .

For two monads,  $(S, \eta^S, \mu^S)$  and  $(T, \eta^T, \mu^T)$ , over the same category, the functor  $TS$  is not necessarily a monad. According to Beck's Theorem [4], the composition of two monads,  $S$  and  $T$ , is a monad if and only if there is a distributive law between them. A distributive law consists of a natural transformation  $\lambda : ST \rightarrow TS$  that satisfies the following equations:

- (i)  $\lambda \circ S\eta^T = \eta^T S$
- (ii)  $\lambda \circ \eta^S T = T\eta^S$
- (iii)  $\lambda \circ S\mu^T = \mu^T S \circ T\lambda \circ \lambda T$
- (iv)  $\lambda \circ \mu^S T = T\mu^S \circ \lambda S \circ S\lambda$

### 5.1 Distributive Law With the Lower Powerdomain

Let  $\Gamma_L$  be the lower powerdomain functor. Suppose  $(M, f) \in RC \circ \Gamma_L(D)$ , so that  $f$  is a function from  $M$  to  $\Gamma_L(D)$ . Now define the natural transformation  $\lambda : RC \circ \Gamma_L(D) \rightarrow \Gamma_L \circ RC(D)$  by:

$$\lambda(M, f) = \downarrow \{(M, g) \mid g(w) \in f(w), \forall w \in M\}$$

**Proposition 5.1** There is a distributive law between the monad of random choice and the lower powerdomain, using the natural transformation  $\lambda$ .

## 5.2 Extending the Monad

The above construction is a monad in the category  $\mathbf{BCD}$ . However, only two of the nondeterministic powerdomains (the upper and lower) leave  $\mathbf{BCD}$  invariant.  $\mathbf{BCD}$  is not closed under the convex powerdomain, but the Cartesian closed categories  $\mathbf{RB}$  and  $\mathbf{FS}$ , which contain  $\mathbf{BCD}$ , are. The monad  $RC$  is not believed to stay within these categories, since we see no way to construct the deflations needed to show that an object is in one or the other of these categories. In  $\mathbf{BCD}$ , infima can be used, but outside of  $\mathbf{BCD}$ , infima are not guaranteed. One way to repair this is to not only define our functions on antichains, but instead to define them on the Scott closure, or lower set, of these antichains. This way, there is no need for infima to project down to smaller trees, since the function is already defined on the lower set.

In our first monad, antichains are used, representing the possible outcomes of a random computation. Now we change this monad to include not only antichains of words, but also the prefixes of these words. These prefixes represent intermediate stages of computation where more random bits are still needed.

**Definition 5.2** A Scott closed set  $M$  in  $\{0,1\}^\infty$  is *full* if for all words,  $w$ , in  $M$ ,  $w * 0 \in M \Leftrightarrow w * 1 \in M$ . Denote the family of nonempty, full Scott closed subsets of  $\{0,1\}^\infty$  by  $\Gamma_f(\{0,1\}^\infty)$ .

$\Gamma_f(\{0,1\}^\infty)$ , ordered by inclusion, is a subposet of the lower powerdomain of  $\{0,1\}^\infty$ . If a poset is a dcpo, domain, or bounded complete domain, then so is the lower powerdomain of that poset. The supremum of some nonempty subset  $\{M_i\}$  is simply the closure of the union,  $\overline{\bigcup_i M_i}$ .

**Definition 5.3** For a category of domains, the functor  $RC'$  is now defined on objects by

$$RC'(D) = \{(M, f) \mid M \in \Gamma_f(\{0,1\}^\infty), f : M \rightarrow D \text{ is Scott continuous}\}$$

For  $a : D \rightarrow D'$  and  $(M, f) \in RC'(D)$ , we define

$$RC'(a)(M, f) = (M, a \circ f)$$

$RC'(D)$  is given an order such that  $(M, f) \sqsubseteq (N, g)$  iff  $M \subseteq N$  and  $f(w) \leq g(w), \forall w \in M$ .

**Theorem 5.4**  $RC'$  is an endofunctor in the categories  $\mathbf{RB}$  and  $\mathbf{FS}$ .

For the monad construction, the unit is the same as before:

$$\eta(d) = (\epsilon, \chi_d)$$

For a continuous function  $h : D \rightarrow RC'(E)$  and some  $(M, f)$  in  $RC'(D)$ , the Kleisli extension is defined by

$$\pi_1 \circ h^\dagger(M, f) = M \cup \left( \bigcup_{w \in M} (\uparrow w \cap (\pi_1 \circ h \circ f(w))) \right)$$

$$((\pi_2 \circ h^\dagger)(M, f))(z) = g(\pi_N(z))$$

where  $(N, g) = h \circ f \circ \pi_M(z)$ .

**Theorem 5.5** *The functor  $RC'$  forms a monad in the categories  $RB$  and  $FS$ .*

### 5.3 Distributive Law With the Convex Powerdomain

Let  $\Gamma_C$  denote the convex powerdomain functor. Recall that the convex powerdomain of a coherent domain  $D$  consists of  $Lens(D)$ , the nonempty lenses of  $D$ . For a nonempty compact  $K \subseteq D$ , define the *lens closure* of  $K$  by  $\langle K \rangle = \overline{K} \cap \uparrow K$ . The lens closure  $\langle K \rangle$  is the smallest lens containing  $K$ .

Suppose  $U \in \Gamma_C \circ RC'(D)$ , so that  $U$  is a lens of random choices of  $X$ . Define the natural transformation,  $\lambda : \Gamma_C \circ RC'(D) \rightarrow RC' \circ \Gamma_C(D)$  by:

$$\lambda(U) = (\overline{\bigcup_{(M,f) \in U} M}, w \mapsto \langle \bigcup_{(M,f) \in U} f \circ \pi_M(w) \rangle)$$

**Proposition 5.6** *There is a distributive law between the monad of random choice and the convex powerdomain, using the natural transformation  $\lambda$ .*

## 6 Relation to Scott's Stochastic Lambda Calculus

Dana Scott developed an operational semantics of the lambda calculus using the power set of the natural numbers,  $\mathcal{P}(\mathbb{N})$ . As terms of the lambda calculus, elements of  $\mathcal{P}(\mathbb{N})$  can be applied to one another and  $\lambda$ -abstraction is achieved through the use of enumerations similar to Gödel numbering.

Scott then added randomness to his model, resulting in his stochastic lambda calculus [20]. He does this by adding random variables.

**Definition 6.1** A *random variable* in Scott's model is a function  $X : [0, 1] \rightarrow \mathcal{P}(\mathbb{N})$  where  $\{t \in [0, 1] \mid n \in X(t)\}$  is Lebesgue measurable for all  $n$  in  $\mathcal{P}(\mathbb{N})$ .

This is similar to the monad of random choice presented in this paper. We start with a base domain  $D$ , which could be  $\mathcal{P}(\mathbb{N})$ , and then have a function from an antichain of  $\{0, 1\}^\omega$  into  $D$ . We can really treat this as a function from the Cantor space,  $\{0, 1\}^\omega$  to  $D$ . For some  $(M, f)$ , define  $\bar{f} : \{0, 1\}^\omega \rightarrow D$  by  $\bar{f}(z) = f(\pi_M(z))$ .

Now that random variables are added to the lambda calculus, there must be a way to define application of one random variable to another. In a sense, this is lifting the application operation from  $\mathcal{P}(\mathbb{N}) \times \mathcal{P}(\mathbb{N})$  to  $([0, 1] \rightarrow \mathcal{P}(\mathbb{N})) \times ([0, 1] \rightarrow \mathcal{P}(\mathbb{N}))$ , which, as stated above, is the role of the Kleisli extension of the monad. Scott defines the application as follows:

**Definition 6.2** Given two random variables  $X, Y : [0, 1] \rightarrow \mathcal{P}(\mathbb{N})$ , the *application operation* is defined by

$$X(Y)(t) = X(t)(Y(t))$$

These random variables can be thought of as using an oracle that randomly gives a element of  $[0, 1]$ , and then the function of the random variable uses this number to output an element of  $\mathcal{P}(\mathbb{N})$ . Notice that in the above definition for application, both random variables receive the same  $t$ . Thus, the oracle is consulted only once

instead of giving a different random number to each random variable. This exactly mimics the concurrent operation of our Kleisli extension. But instead of an oracle giving an entire real number at once (which has infinite information), the oracle gives one bit at a time.

## 7 Summary and Future Work

In this paper, we have presented two monads for randomized computation in Cartesian closed categories of domains. Computational motivation is given for the structure of these monads. In a program, random choice results in the branching of computation, so the possible outcomes form a tree. Our first monad separates random choice from the underlying domain and confines it to the leaves of a binary tree. This is the main difference between our construction and the probabilistic powerdomain. We have shown that this monad captures the randomized behavior found in algorithms such as the Miller-Rabin primality test. We have given a new Kleisli extension that satisfies the monad laws and presented a distributive law with the lower powerdomain, all within the category  $\mathbf{BCD}$ . In order to work with the convex powerdomain, we needed to move into the category  $\mathbf{RB}$  or  $\mathbf{FS}$ . A slight change was needed for our construction to stay within these categories, and a distributive law was given between this extended monad and the convex powerdomain.

There is much work to be done concerning these monads. Some work has already been completed that is beyond the scope of this paper. Another alteration can be made to the monad to obtain a distributive law with the upper powerdomain. Furthermore, an operational version of the monad has been developed and implemented in functional programming languages such as Scala and Haskell. The proof that the monad laws hold for this operational version has been formally verified using Isabelle. Finally, Randomized PCF (rPCF), a programming language that adds random choice to PCF [19], has been designed, and the Miller-Rabin algorithm has been implemented within the language. An operational and denotational semantics for rPCF have been developed using the monad presented in this paper. This is a proof of concept to show how this monad can be used to augment other languages with random choice.

## 8 Acknowledgements

The author thanks Michael Mislove for the guidance and the many fruitful discussions that made this work possible, along with several much-needed suggestions for the preparation of this paper. The author also thanks Jean Goubault-Larrecq and Dana Scott for email correspondence regarding the model of continuous random variables and the stochastic lambda calculus, respectively. Finally, the author acknowledges the support of the AFOSR under award no. FA0550-13-1-0135 during the preparation of this work.

## References

- [1] Abramsky, S. and A. Jung, *Domain theory*, Handbook of logic in computer science **3** (1994), pp. 1–168.

- [2] Abramsky, S. and N. Tzevelekos, *Introduction to categories and categorical logic*, in: *New structures for physics*, Springer, 2011 pp. 3–94.
- [3] Agrawal, M., N. Kayal and N. Saxena, *Primes is in P*, Annals of mathematics (2004), pp. 781–793.
- [4] Beck, J., *Distributive laws*, in: *Seminar on triples and categorical homology theory*, Springer, 1969, pp. 119–140.
- [5] De Leeuw, K., E. F. Moore, C. E. Shannon and N. Shapiro, *Computability by probabilistic machines*, Automata studies **34** (1956), pp. 183–198.
- [6] Division, I. B. M. C. R. and M. Rabin, “Probabilistic algorithms,” 1976.
- [7] Gierz, G., K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove and D. S. Scott, *Continuous lattices and domains, volume 93 of encyclopedia of mathematics and its applications* (2003).
- [8] Gill, J., *Computational complexity of probabilistic turing machines*, SIAM Journal on Computing **6** (1977), pp. 675–695.
- [9] Goubault-Larrecq, J. and D. Varacca, *Continuous random variables*, in: *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science (LICS’11)* (2011), pp. 97–106.  
URL <http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/GLV-lics2011.pdf>
- [10] Jung, A., “Cartesian closed categories of domains,” Citeseer, 1989.
- [11] Jung, A. and R. Tix, *The troublesome probabilistic powerdomain*, Electronic Notes in Theoretical Computer Science **13** (1998), pp. 70–91.
- [12] Mislove, M., *Nondeterminism and probabilistic choice: Obeying the laws*, in: *CONCUR 2000 Concurrency Theory*, Springer, 2000 pp. 350–365.
- [13] Mislove, M., *Discrete random variables over domains*, Theoretical computer science **380** (2007), pp. 181–198.
- [14] Mislove, M., *Anatomy of a domain of continuous random variables ii*, in: *Computation, Logic, Games, and Quantum Foundations. The Many Facets of Samson Abramsky*, Springer, 2013 pp. 225–245.
- [15] Mislove, M. W., *Topology, domain theory and theoretical computer science*, Topology and its Applications **89** (1998), pp. 3–59.
- [16] Moggi, E., *Notions of computation and monads*, Information and computation **93** (1991), pp. 55–92.
- [17] Rabin, M. O., *Probabilistic algorithm for testing primality*, Journal of number theory **12** (1980), pp. 128–138.
- [18] Saheb-Djahromi, N., *Cpo’s of measures for nondeterminism*, Theoretical Computer Science **12** (1980), pp. 19–37.
- [19] Scott, D. S., *A type-theoretical alternative to ISWIM, CUCH, OWHY*, Theoretical Computer Science **121** (1993), pp. 411–440.
- [20] Scott, D. S., *Stochastic  $\lambda$ -calculi*, Journal of Applied Logic **12** (2014), pp. 369–376.
- [21] Solovay, R. and V. Strassen, *A fast monte-carlo test for primality*, SIAM journal on Computing **6** (1977), pp. 84–85.
- [22] Tix, R., “Continuous D-cones: convexity and powerdomain constructions,” Shaker, 1999.
- [23] Tix, R., K. Keimel and G. Plotkin, *Semantic domains for combining probability and non-determinism*, Electronic Notes in Theoretical Computer Science **222** (2009), pp. 3–99.
- [24] Varacca, D., “Probability, nondeterminism and concurrency: two denotational models for probabilistic computation,” BRICS, 2003.
- [25] Varacca, D., G. Winskel et al., *Distributing probability over non-determinism*, Mathematical Structures in Computer Science **16** (2006), pp. 87–113.



# Binding Operators for Nominal Sets

Arthur Azevedo de Amorim<sup>1</sup>

*Department of Computer and Information Science  
University of Pennsylvania  
Philadelphia, PA, United States*

---

## Abstract

The theory of nominal sets is a rich mathematical framework for studying syntax and variable binding. Within it, we can describe several binding disciplines and derive convenient reasoning principles that respect  $\alpha$ -equivalence. In this article, we introduce the notion of *binding operator*, a novel construction on nominal sets that unifies and generalizes many forms of binding proposed in the literature. We present general results about these operators, including sufficient conditions for validly using them in inductive definitions of nominal sets.

*Keywords:* Nominal Sets, Binding, Alpha Equivalence

---

## 1 Introduction

Bound variables have puzzled computer scientists and logicians for decades. Although fairly simple to handle in informal pencil-and-paper calculations, they can be surprisingly complex to manage in algorithms and mechanized proofs, where the mostly uninteresting formal details of variable binding cannot be overlooked. Research on the subject has led to various promising approaches for tackling this complexity [6,12,14], among which we can mention the theory of *nominal sets* [5].

Nominal sets constitute a rich mathematical universe where objects contain *variables* that can be *renamed*, allowing various notions of  $\alpha$ -equivalence to be defined. In the  $\lambda$ -calculus for example, we stipulate that the term  $\lambda x. t$  is equivalent to any other obtained by renaming  $x$  to a variable  $y$  that does not appear free in  $t$ , which corresponds to the operation of *name abstraction* on nominal sets [5], used for modeling objects with a single bound variable. The nominal literature has shown how many other forms of binding can be obtained through similar constructions, such as *generalized* name abstractions [4,3], or the binding declarations of Nominal Isabelle [17]. Besides serving as a good theoretical foundation for variable binding,

---

<sup>1</sup> The author thanks Andrew Pitts, Randal Clouston, and Matthew Weaver for comments on early drafts of this article.

nominal techniques have influenced the design of many tools for manipulating syntax, such as the FreshML programming language [11,15] and the Nominal package for Isabelle/HOL [16,17].

Although some of the notions above are more general than others, none of them proposes to offer a clear, unified picture of what binding means for nominal sets. In this article, we attempt to look at the problem from a more foundational perspective, by introducing *binding operators*: a novel construction on nominal sets that unifies and generalizes many forms of variable binding proposed in the literature. After briefly recalling basic notions of nominal set theory (Section 2), we introduce binding operators in Section 3, showing how to use them for defining a variety of nominal sets representing binders in Section 4. Section 5 gives an alternative characterization of these nominal sets defined by binding operators, used in Section 6 to encompass variable scope within our framework. In Section 7, we discuss category-theoretic properties of binding operators, which provide sufficient conditions for defining nominal sets inductively. We conclude and review related work in Section 8.

## 2 Preliminaries: Nominal Sets

We begin by recalling basic concepts and results of the theory of nominal sets; for a detailed account on the subject, we refer the reader to the introductory article by Gabbay and Pitts [5] or to Pitts' book [10].

We fix some countably infinite set  $\mathbb{A}$ . We refer to elements of  $\mathbb{A}$  as *atoms*, and use the variable  $a$  to denote them. A *permutation* of  $\mathbb{A}$  is a bijective function  $\pi : \mathbb{A} \rightarrow \mathbb{A}$  such that  $\pi(a) = a$  for all but finitely many  $a \in \mathbb{A}$ . Permutations form a group under composition, noted  $\text{perm}(\mathbb{A})$ ; in particular,  $\pi \circ \pi' \in \text{perm}(\mathbb{A})$  and  $\pi^{-1} \in \text{perm}(\mathbb{A})$  for every  $\pi, \pi' \in \text{perm}(\mathbb{A})$ .

A *renaming operation* on a set  $X$  is a *group action* of  $\text{perm}(\mathbb{A})$  on  $X$ . Spelled out explicitly, this means a mapping that to each pair  $(\pi, x) \in \text{perm}(\mathbb{A}) \times X$  associates an element  $\pi \cdot x \in X$ , so that

$$1 \cdot x = x \qquad (\pi_1 \circ \pi_2) \cdot x = \pi_1 \cdot \pi_2 \cdot x,$$

where  $1 \in \text{perm}(\mathbb{A})$  denotes the identity function. We treat renaming as right associative, reading  $\pi_1 \cdot \pi_2 \cdot x$  as  $\pi_1 \cdot (\pi_2 \cdot x)$ . The above properties imply in particular  $\pi^{-1} \cdot \pi \cdot x = \pi \cdot \pi^{-1} \cdot x = x$  for arbitrary  $\pi$  and  $x$ .

We say that a set of atoms  $A$  *supports* an element  $x \in X$  if the atoms in  $A$  completely determine the effect of renaming on  $x$ . Formally, if  $\pi$  is a permutation such that  $\pi(a) = a$  for every  $a$  in  $A$ , then  $\pi \cdot x = x$ . Or, equivalently, if  $\pi_1$  and  $\pi_2$  are permutations such that  $\pi_1(a) = \pi_2(a)$  for every  $a$  in  $A$ , then  $\pi_1 \cdot x = \pi_2 \cdot x$ . If  $A$  is finite, we can show [5] that  $x$  has a *minimal* finite supporting set  $\text{supp}(x)$ , by which we mean that  $\text{supp}(x)$  is a subset of every finite set  $A'$  supporting  $x$ . We say that  $X$  is a *nominal set* if all of its elements have finite support.

Atoms  $\mathbb{A}$  form a nominal set under the action  $\pi \cdot a = \pi(a)$ , with  $\text{supp}(a) = \{a\}$ . We can see every set  $X$  as a trivial nominal set by posing  $\pi \cdot x = x$ , which implies  $\text{supp}(x) = \emptyset$ . We use this structure for sets such as  $\mathbb{N}$  or  $\mathbb{Z}$ , whose elements

intuitively do not contain variables. A nominal set with such a trivial renaming operation is called *discrete*. We can also define products and disjoint unions of nominal sets, summarized in the table below.

$X_1 \times X_2$	$\pi \cdot (x_1, x_2) = (\pi \cdot x_1, \pi \cdot x_2)$	$\text{supp}(x_1, x_2) = \text{supp}(x_1) \cup \text{supp}(x_2)$
$X_1 + X_2$	$\pi \cdot (i, x_i) = (i, \pi \cdot x_i)$	$\text{supp}(i, x_i) = \text{supp}(x_i)$

When working with nominal sets, we want to restrict our attention to functions that are well-behaved with respect to renaming. A function  $f : X \rightarrow Y$  between nominal sets is said to be *equivariant* if it commutes with renaming; that is,  $f(\pi \cdot x) = \pi \cdot f(x)$  for every  $x$  and  $\pi$ . We write  $X \rightarrow_{\text{eq}} Y$  for the set of equivariant functions from  $X$  to  $Y$ . When  $Y$  is the discrete nominal set of booleans  $\mathbb{B}$ , we sometimes say that  $f$  is an equivariant property or relation instead. This is equivalent to saying that  $f(\pi \cdot x)$  holds if and only if  $f(x)$  does. Every such property can be alternatively seen as a *nominal subset* of  $X$ ; that is, a subset of  $X$  that is closed under renaming. Note that equivariant functions cannot add atoms to the support of their arguments: we can show that  $\text{supp}(f(x)) \subseteq \text{supp}(x)$ , with  $\text{supp}(f(x)) = \text{supp}(x)$  if  $f$  is injective.

The next best thing to an equivariant function is a *finitely supported* one: a function  $f$  between nominal sets  $X$  and  $Y$  that is *almost* equivariant, *except* for a finite set of atoms  $A$ ; that is,  $f(\pi \cdot x) = \pi \cdot f(x)$  if  $\pi(a) = a$  for every  $a \in A$ . We write  $X \rightarrow_{\text{fs}} Y$  for the set of finitely supported functions from  $X$  to  $Y$ . Every equivariant function is trivially finitely supported. Finitely supported functions  $f$  form a nominal set under the action  $(\pi \cdot f)(x) = \pi \cdot f(\pi^{-1} \cdot x)$ . This is equivalent to saying that  $(\pi \cdot f)(\pi \cdot x) = \pi \cdot f(x)$  for every  $\pi$  and  $x$ , which allows us to depict this renaming operation as acting on a table representation of  $f$ :

$$\begin{array}{ccc} x_1 \mapsto f(x_1) & & \pi \cdot x_1 \mapsto \pi \cdot f(x_1) \\ x_2 \mapsto f(x_2) & \xrightarrow{\pi} & \pi \cdot x_2 \mapsto \pi \cdot f(x_2) \\ \vdots & & \vdots \end{array}$$

Note that the support of a function is not computable in general. We use similar actions for other sets of functions; for instance,  $\text{perm}(\mathbb{A})$  is a nominal set under the action  $\pi \cdot \pi' = \pi \circ \pi' \circ \pi^{-1}$ , with  $\text{supp}(\pi) = \{a \mid \pi(a) \neq a\}$ . Seeing a subset  $X' \subseteq X$  as a function  $X \rightarrow \mathbb{B}$  results in a renaming operation defined by  $\pi \cdot X' = \{\pi \cdot x \mid x \in X'\}$ .

Let  $X$  and  $Y$  be two nominal sets, and  $x$  and  $y$  be elements of  $X$  and  $Y$ . We say that  $x$  and  $y$  are fresh with respect to each other, noted  $x \# y$ , if their supports are disjoint:  $\text{supp}(x) \cap \text{supp}(y) = \emptyset$ . If  $a \in \mathbb{A}$  and  $x \in X$ , then  $a \# x$  simply means that  $a \notin \text{supp}(x)$ . If  $\pi \in \text{perm}(\mathbb{A})$ ,  $\pi \# x$  is equivalent to  $\pi(a) = a$  for every  $a \in \text{supp}(x)$ , which implies  $\pi \cdot x = x$ . In particular, if  $f$  is a finitely supported function,  $\pi \# f$  implies  $f(\pi \cdot x) = \pi \cdot f(x)$  for every  $x$ .

Nominal sets and equivariant functions between them form a category **Nom**. It is a complete and cocomplete category; in particular, the initial and terminal objects are the empty and singleton discrete nominal sets, while binary products and sums are given as in the above table. It is also a cartesian closed category, with exponentials given by finitely supported functions.

### 3 Binding Operators

The most basic form of binding on nominal sets is *name abstraction* [5]. Given a nominal set  $X$ , we define an equivalence relation  $\equiv_\alpha$  on  $\mathbb{A} \times X$  by saying that  $(a_1, x_1) \equiv_\alpha (a_2, x_2)$  if and only if

$$\exists a_3. a_3 \# (a_1, a_2, x_1, x_2) \wedge (a_1 a_3) \cdot x_1 = (a_2 a_3) \cdot x_2 \quad (1)$$

Here,  $(aa')$  denotes the *transposition* of  $a$  and  $a'$ , which swaps these two atoms while fixing all others. Intuitively, this relation states that  $a$  is bound in the pair  $(a, x)$  and should be treated up to  $\alpha$ -equivalence. If we quotient  $\mathbb{A} \times X$  by this relation, we obtain a new nominal set  $[\mathbb{A}]X$ , called the set of name abstractions of  $X$ , where  $\alpha$ -equivalent objects become equal.

Besides name abstraction, we can define nominal sets for representing many other binding disciplines, such as name restriction or ML's `let rec`. A common feature of these constructions is that equivalent elements are obtained by renaming bound atoms while fixing those that remain free. For instance, although not immediately obvious, we can rephrase (1) as

$$(a_1, x_1) \equiv_\alpha (a_2, x_2) \iff \exists \pi. \pi \# \text{supp}(x_1) \setminus \{a_1\} \wedge (a_2, x_2) = \pi \cdot (a_1, x_1). \quad (2)$$

Recall that  $\pi \# \text{supp}(x_1) \setminus \{a_1\}$  simply means that  $\pi$  fixes all elements of that set. If we interpret the singleton  $\{a_1\}$  in this formula as the set of bound variables of the pair  $(a_1, x_1)$ , we get a generic method for defining  $\alpha$ -equivalence for other binders: it suffices to enumerate which atoms should be bound in an object. Formally, we have the following definition.

**Definition 3.1** Let  $X$  be a nominal set. A *binding operator* on  $X$  is an equivariant function  $l : X \rightarrow_{\text{eq}} \mathcal{P}_{\text{fin}}(\mathbb{A})$ . Each  $l$  gives rise to a relation  $\equiv_l$  on  $X$ , defined as

$$x_1 \equiv_l x_2 \iff \exists \pi. \pi \# \text{supp}(x_1) \setminus l(x_1) \wedge x_2 = \pi \cdot x_1.$$

Thus, we see that  $\alpha$ -equivalence for name abstractions corresponds to a binding operator on  $\mathbb{A} \times X$ , defined as  $l_\alpha(a, x) = \{a\}$ . We analyze other examples in Section 4, but need to explain first how exactly binding operators are used to encode binders as nominal sets. Concretely, we show here that every binding operator induces a quotient nominal set, a direct generalization of the analogous results for name abstractions. We begin by noting the following simple facts.

**Lemma 3.2** Let  $X$  be a nominal set with a binding operator  $l$ . If  $x_1 \equiv_l x_2$ , then  $\text{supp}(x_1) \setminus l(x_1) = \text{supp}(x_2) \setminus l(x_2)$ .

**Proof** The definition implies that  $x_2$  is of the form  $\pi \cdot x_1$ , with  $\pi \# \text{supp}(x_1) \setminus l(x_1)$ ; thus,  $\pi \cdot (\text{supp}(x_1) \setminus l(x_1)) = \text{supp}(x_1) \setminus l(x_1)$ . The result then follows by equivariance, since the right-hand side is equal to  $\text{supp}(\pi \cdot x_1) \setminus l(\pi \cdot x_1) = (\pi \cdot \text{supp}(x_1)) \setminus (\pi \cdot l(x_1)) = \pi \cdot (\text{supp}(x_1) \setminus l(x_1))$ .  $\square$

**Lemma 3.3** Let  $X$  be a nominal set endowed with a binding operator  $l$ . The  $\equiv_l$  relation is an equivariant equivalence relation.

**Proof** The relation is clearly reflexive: it suffices to take  $\pi = 1$  in its definition. It is also equivariant, because it is defined with equivariant operations.

To see that it is symmetric, take two elements  $x$  and  $x'$  of  $X$  such that  $x \equiv_l x'$ . By definition, we can find  $\pi \in \text{perm}(\mathbb{A})$  such that  $\pi \# \text{supp}(x) \setminus l(x)$  and  $x' = \pi \cdot x$ . We must show that  $\pi \cdot x \equiv_l x$ . Since  $x = \pi^{-1} \cdot \pi \cdot x$ , it suffices to show that

$$\pi^{-1} \# \text{supp}(\pi \cdot x) \setminus l(\pi \cdot x),$$

which holds by equivariance, because  $\pi^{-1} = \pi \cdot \pi^{-1} = \pi \circ \pi^{-1} \circ \pi^{-1}$ .

Finally, let's show transitivity. Take three elements,  $x_1$ ,  $x_2$  and  $x_3$ , such that  $x_1 \equiv_l x_2$  and  $x_2 \equiv_l x_3$ . By unfolding definitions, and using Lemma 3.2, we find  $\pi_1$  and  $\pi_2$  such that  $x_3 = (\pi_2 \circ \pi_1) \cdot x_1$  and  $\pi_i \# (\text{supp}(x_1) \setminus l(x_1))$  for  $i = 1, 2$ . Since permutation composition is equivariant, we see that  $\text{supp}(\pi_2 \circ \pi_1) \subseteq \text{supp}(\pi_1) \cup \text{supp}(\pi_2)$ ; thus, the freshness conditions above yield  $\pi_2 \circ \pi_1 \# \text{supp}(x_1) \setminus l(x_1)$ , allowing us to conclude.  $\square$

Because binding operators yield equivariant equivalence relations, they lead to quotients that carry a canonical nominal structure:

**Lemma 3.4** *Let  $X$  be a nominal set with a binding operator  $l$ , and let  $X/l$  be the quotient of  $X$  by the equivalence relation  $\equiv_l$ . This set possesses a nominal structure satisfying*

$$\pi \cdot [x] = [\pi \cdot x] \quad \text{supp}([x]) = \text{supp}(x) \setminus l(x),$$

where  $[x]$  denotes the equivalence class of  $x$  under  $\equiv_l$ . In particular, the canonical projection into  $X/l$  is equivariant.

**Proof** Any quotient by an equivariant equivalence relation carries a canonical nominal structure satisfying the first identity [10, Sections 1.8 and 2.9]. We also have [10, Proposition 2.30]

$$\text{supp}([x]) = \bigcap_{x' \equiv_l x} \text{supp}(x'). \quad (3)$$

By Lemma 3.2, the right-hand side equals

$$\text{supp}(x) \setminus l(x) \cup \bigcap_{x' \equiv_l x} l(x').$$

We can conclude because the second term of the union is empty. More precisely, given any atom  $a$  in  $l(x)$ , and any atom  $a'$  that is not in  $\text{supp}(x)$ , we have  $(a a') \cdot x \equiv_l x$ , but  $a = (a a') \cdot a'$  is not in  $l((a a') \cdot x)$  by equivariance.  $\square$

As a sanity check, if we instantiate the previous result with  $l_\alpha$ , the binding operator for name abstractions, we obtain the familiar identities  $\pi \cdot \langle a \rangle x = \langle \pi(a) \rangle (\pi \cdot x)$  and  $\text{supp}(\langle a \rangle x) = \text{supp}(x) \setminus \{a\}$ , where  $\langle a \rangle x$  denotes the equivalence class of the pair  $(a, x)$  in  $[\mathbb{A}]X$ . By virtue of being defined as a quotient, we also obtain generic *elimination principles* for such nominal sets. Equivariant functions on them have a particularly simple characterization: they correspond to functions that do not leak bound atoms in their results.

**Lemma 3.5** *Let  $X$  and  $Y$  be nominal sets, and  $l$  be a binding operator on  $X$ . Let  $f : X \rightarrow_{\text{eq}} Y$  be a function satisfying  $l(x) \# f(x)$  for all  $x$ . There exists a unique  $\bar{f} : X/l \rightarrow_{\text{eq}} Y$  such that  $\bar{f}([x]) = f(x)$  for all  $x$ . Conversely, every function  $f : X \rightarrow_{\text{eq}} Y$  that factors through  $X/l$  satisfies  $l(x) \# f(x)$ .*

**Proof** To build  $\bar{f}$ , it suffices to show that, for every  $x_1 \equiv_l x_2$ , we have  $f(x_1) = f(x_2)$ . By the definition of  $\equiv_l$ , we find a permutation  $\pi$  that is fresh for  $\text{supp}(x_1) \setminus l(x_1)$  such that  $x_2 = \pi \cdot x_1$ . Thus,  $f(x_2) = \pi \cdot f(x_1)$ . Since  $l(x_1) \# f(x_1)$ , it must be the case that  $\text{supp}(f(x_1)) \subseteq \text{supp}(x_1) \setminus l(x_1)$ . This implies that  $\pi \# f(x_1)$ , and thus  $f(x_1) = \pi \cdot f(x_1) = f(x_2)$ . The last assertion follows because, if  $f(x) = \bar{f}([x])$  for some equivariant function  $\bar{f}$ , then  $\text{supp}(f(x))$  is contained in  $\text{supp}([x])$ , which equals  $\text{supp}(x) \setminus l(x)$ .  $\square$

Later, in Lemma 6.14, we extend this result to describe the finitely supported functions that can be defined on such quotients. It would be possible (and not too difficult) to state this extension right away and prove it directly, but the tools developed in Section 6 provide more structure for attacking the problem.

Notice that the above properties only rely on knowing which atoms are bound in an object, and how these atoms are affected by renaming. This is indeed the only piece of information that we can extract from binding operators, which hide everything else that we might care about in bound atoms—for instance, the order in which they appear. Fortunately, as shown in the rest of this paper, this extra information is irrelevant for deriving the fundamental properties of binding constructs in nominal sets.

## 4 Examples

Given the generality of binding operators, it is worth analyzing a few examples of binding disciplines that they can express. We show here how to model a few syntactic constructs that have been extensively studied in the literature. We include an example of common idiom that is not directly supported by our framework—namely, binding atoms in only *part* of an object. Fortunately, as shown in Section 6, this limitation is not fundamental, and can be overcome by adding a notion of scope to binding operators.

### 4.1 Generalized Name Abstraction

Name abstraction binds a single atom within an object. The simplest way to generalize it is to consider constructions where bound atoms are specified by arbitrary data structures that contain atoms, leading to so-called *generalized* name abstractions [4]. Specifically, given nominal sets  $X$  and  $Y$ , we adapt the binding operator  $l_\alpha$  defining name abstraction to  $X \times Y$ , by setting  $l_\alpha(x, y) = \text{supp}(x)$ . The corresponding quotient, noted  $[X]Y$ , is known as the nominal set of  $X$ -abstractions of  $Y$ .

#### 4.2 Name Restriction

Processes in the  $\pi$ -calculus [8] communicate through named channels, which can be made private using a form of binding known as *name restriction*. Concretely, an expression of the form  $\nu a. t$  denotes a computation  $t$  that has access to a communication channel  $a$  bound by  $\nu$ , which cannot be used by any other processes defined outside of this expression.

Besides being subject to  $\alpha$ -equivalence of bound channel names,  $\pi$ -calculus processes satisfy certain behavioral identities related to name restriction. For instance, the order in which channels are bound in a process expression is irrelevant:

$$\nu a_1. \nu a_2. t = \nu a_2. \nu a_1. t.$$

To represent name restriction, we could be tempted to model  $\pi$ -calculus terms with a nominal set of the form  $[\mathcal{P}_{\text{fin}}(\mathbb{A})]E$ , where  $E$  denotes a nominal set of process expressions. The idea is that an expression of the form  $\nu a_1. \dots \nu a_n. t$  would correspond to the element  $[(\{a_1, \dots, a_n\}, t)]$ . Unfortunately, this encoding does not validate another basic property of name restriction: spurious private channels do not affect process behavior. Formally, if  $a$  does not occur free in  $t$ , then  $\nu a. t = t$ .

We solve this problem by restricting our binding operator to a nominal subset of  $\mathcal{P}_{\text{fin}}(\mathbb{A}) \times E$  that excludes spurious atoms. Specifically, we pose

$$L(E) = \{(A, t) \in \mathcal{P}_{\text{fin}}(\mathbb{A}) \times E \mid A \subseteq \text{supp}(t)\},$$

and quotient this nominal set by the binding operator  $l(A, t) \triangleq A$ . The resulting nominal set, noted  $\text{Res}(E)$ , is known as the *free nominal restriction set* over  $E$  [10, Chapter 9]. We can then represent an expression  $\nu a_1. \dots \nu a_n. t$  by the element  $[(\{a_1, \dots, a_n\} \cap \text{supp}(t), t)]$ . Besides their application to the  $\pi$ -calculus, free nominal restriction sets yield a monad on  $\text{Nom}$  that provides an useful model of fresh-name generation.

#### 4.3 Mutually Recursive Definitions

Most programming languages allow mutually recursive function definitions. In the ML family, these usually take the form

$$\text{let rec } a_1 = t_1 \text{ and } \dots \text{ and } a_n = t_n \text{ in } t,$$

where the atoms  $a_1, \dots, a_n$  are bound in the expressions  $t, t_1, \dots, t_n$ . We could represent mutually recursive definitions with a nominal set of the form  $[\text{List}(\mathbb{A})] \text{List}(E)$ , where  $E$  is some nominal set of expressions, and  $\text{List}(X)$  is the nominal set of finite lists of elements of  $X$ . The idea is that an expression such as the one above would be mapped to the element  $[(a_1, \dots, a_n), [t_1, \dots, t_n, t]]$ . The problem, as noted by Pottier [12], is that this nominal set contains elements that do not correspond to any valid expression, such as  $[(a), []]$ , where the number of defined atoms does not match the number of definition bodies.

We can use binding operators to model mutually recursive definitions by viewing expressions as the one above as a pair  $(f, t)$ , where  $t \in E$  is an expression, and

$f : \mathbb{A} \rightarrow_{\text{fin}} E$  is a partial function with finite domain. The term  $t$  represents the result of the expression, while the function  $f$  maps each atom to the corresponding definition; in the example given above, this would be a function mapping  $a_1$  to  $t_1$ ,  $a_2$  to  $t_2$ , etc. (Note that this assumes that the order of the definitions does not matter. We could also have considered a more concrete variant with lists of declarations that have an explicit order.) The bound atoms in  $(f, t)$  are exactly those in the domain of  $f$ , which leads to the nominal set  $\text{Mut}(E) \triangleq ((\mathbb{A} \rightarrow_{\text{fin}} E) \times E) / (\text{dom} \circ p_1)$ , where  $p_1$  designates the first projection function. This solution is similar to others proposed in the literature [12, 17].

#### 4.4 An Obstacle: Binder Scope

We could try to adapt the previous example to model parallel nonrecursive definitions:

$$\text{let } a_1 = t_1 \text{ and } \dots \text{ and } a_n = t_n \text{ in } t,$$

where the atoms  $a_1, \dots, a_n$  are bound in  $t$ , but not in  $t_1, \dots, t_n$ . Unfortunately, the tools that we have developed so far cannot take this form of scoped binding into account, because the equivalence derived from binding operators require atoms to be renamed *everywhere*, including in positions where they should remain free (in this case, the  $t_i$ ). We will see later, in Section 6, how to work around this issue by considering renaming operations that act only on a limited scope within an object.

## 5 Binding Functions

It is basic set theory that every surjective function  $f$  corresponds to a quotient by an equivalence relation—namely, the one where  $x_1$  and  $x_2$  are equivalent if and only if  $f(x_1) = f(x_2)$ . In this section, we prove an analogous result for binding operators, showing that their quotients can alternatively be characterized as what we call *binding functions*. This characterization will be useful in Section 6, where we use it to relate a more general class of quotients on nominal sets to binding operators.

**Definition 5.1** An equivariant function  $f$  between nominal sets is a *binding function* if it is surjective and, whenever  $f(x_1) = f(x_2)$ , we have  $x_1 \equiv_{l_f} x_2$ , where  $l_f(x) = \text{supp}(x) \setminus \text{supp}(f(x))$ .<sup>2</sup> This last condition simply means that there exists a permutation  $\pi$ , with  $\pi \# f(x_1)$ , such that  $x_2 = \pi \cdot x_1$ .

Intuitively, a binding function is one that removes atoms from the support of its argument, but doesn't discard any other information attached to it. Notice that, by construction, the projection into a quotient by a binding operator is a binding function. However, the converse also holds: every binding function corresponds to a quotient by a binding operator. More precisely, we have the following result.

**Lemma 5.2** *If  $f : X \rightarrow_{\text{eq}} Y$  is a binding function, then there is an isomorphism  $i : Y \cong X / l_f$  such that  $i(f(x)) = [x]$ . In other words,  $f$  is a coequalizer of the equivalence relation  $\equiv_{l_f}$ .*

<sup>2</sup> Kurz et al. [7, Notation 5.40] refer to this binding operator as the set of “bound variables relative to a map”, and use it to study variable binding in infinite objects.



**Proof** We already know that  $f$  is surjective and that  $f(x_1) = f(x_2)$  implies  $x_1 \equiv_{l_f} x_2$ . Conversely, we can see that  $x_1 \equiv_{l_f} x_2$  implies  $f(x_1) = f(x_2)$ , since  $f(x_1) = f(\pi \cdot x_1)$  when  $\pi \# f(x_1)$ . This proves that  $f$  is the coequalizer we're looking for.  $\square$

Binding operators can be combined by composing their quotients:

**Lemma 5.3** *If  $f$  and  $g$  are binding, then so is  $h = gf$ , and  $l_h(x) = l_f(x) \cup l_g(f(x))$ .*

**Proof** It is clear that  $h$  is surjective, as the composition of two surjections. Now, suppose that  $g(f(x_1)) = g(f(x_2))$ . Since  $g$  is binding, we find a permutation  $\pi$ , with  $\pi \# g(f(x_1))$ , such that  $f(x_2) = \pi \cdot f(x_1) = f(\pi \cdot x_1)$ . But  $f$  is also binding, so we get another permutation  $\pi'$  such that  $\pi' \# f(\pi \cdot x_1)$  and  $x_2 = \pi' \cdot \pi \cdot x_1$ . The freshness assumptions on  $\pi$  and  $\pi'$  imply that  $\pi' \circ \pi \# g(f(x_1))$ , because

$$\text{supp}(g(f(x_1))) = \text{supp}(g(f(\pi \cdot x_1))) \subseteq \text{supp}(f(\pi \cdot x_1)).$$

This shows that  $h$  is binding. The last claim follows because

$$\begin{aligned} l_h(x) &= \text{supp}(x) \setminus \text{supp}(g(f(x))) \\ &= (\text{supp}(x) \setminus \text{supp}(f(x))) \cup (\text{supp}(f(x)) \setminus \text{supp}(g(f(x)))) \\ &= l_f(x) \cup l_g(f(x)). \end{aligned}$$

$\square$

Finally, as an aside, we note that every equivariant function can be factored through a quotient by a binding operator. We can compare this result to the more familiar one that says that every function can be factored through its image.

**Lemma 5.4** *Let  $f : X \rightarrow_{\text{eq}} Y$  be an equivariant function. We can factor  $f$  through  $X/l_f$  as  $f = \bar{f} \circ [-]$ :*

$$X \twoheadrightarrow X/l_f \xrightarrow{\bar{f}} Y,$$

where  $\bar{f}$  preserves supports, in the following sense:

$$\text{supp}(\bar{f}(\bar{x})) = \text{supp}(\bar{x}).$$

Furthermore, this factorization is unique up to isomorphism. Specifically, if  $f = hg$ , where  $g$  is binding and  $h$  preserves supports, there exists an isomorphism  $i$  such that the following diagram commutes:

$$\begin{array}{ccc} X & \twoheadrightarrow & X/l_f \\ \downarrow g & \nearrow i & \downarrow \bar{f} \\ Y' & \xrightarrow{h} & Y \end{array}$$

**Proof** That  $f$  can be factored through  $X/l_f$  follows from Lemma 3.5. Because  $h$  preserves supports, we find that  $l_g = l_f$ , and construct  $i$  using Lemma 5.2.  $\square$

The analogy with the image of a function goes even further: we can show that binding functions and functions that preserve supports form a *factorization system* [1, Definition 5.5.1] on the category of nominal sets. Spelled out in detail, this

means that both classes of functions contain all isomorphisms, are closed under composition, and can be used to factor uniquely (up to isomorphism) any equivariant function, as shown above.

## 6 Atom Scope and Freshening

In Section 4.4, we noted that binding operators cannot express syntactic forms that bind atoms in only part of an object. We show here how to accommodate such constructs by decomposing the renaming operation of a nominal set into smaller independent ones. The idea is that each of these independent operations applies an atom permutation to part of an object without affecting the rest, thus allowing bound atoms to  $\alpha$ -vary within their intended scope. The corresponding quotients are *not* binding functions in the sense of Definition 5.1, but we show here that they still support similar elimination principles to those obtained from Lemma 3.5. Besides allowing us to model more binders, this machinery will be useful for deriving stronger elimination principles that work with finitely supported functions.

### 6.1 Independence

The main technical device that we need is the notion of *independence* of two renaming operations.

**Definition 6.1** Let  $X$  be a set with two renaming operations,  $\odot_1$  and  $\odot_2$ . We say that these operations are *independent* if they commute, in the following sense:

$$\pi_1 \odot_1 \pi_2 \odot_2 x = \pi_2 \odot_2 \pi_1 \odot_1 x.$$

We use the  $\odot$  operator to denote a set of multiple independent renaming operations on a set, whereas  $\cdot$  is reserved to its canonical nominal structure. If the elements of  $X$  are finitely supported with respect to these operations, we say that  $X$  has two independent nominal structures. We note  $\text{supp}_1(x)$  and  $\text{supp}_2(x)$  the supports of an element  $x$  of  $X$  with respect to each of the renaming operations.

As an example, if  $X$  and  $Y$  are nominal sets, we can define two independent renaming operations on the product  $X \times Y$  by posing

$$\pi \odot_1 (x, y) \triangleq (\pi \cdot x, y) \qquad \pi \odot_2 (x, y) \triangleq (x, \pi \cdot y).$$

Note that we can express the product nominal set  $X \times Y$  as the composition of these two operations. As a matter of fact, any set with two independent renaming operations can be endowed with a compound one, as shown in the following results.

**Lemma 6.2** *Let  $X$  be a set with two independent nominal structures. The support of an element with respect to one structure is invariant with respect to the other:*

$$\text{supp}_1(\pi \odot_2 x) = \text{supp}_1(x) \qquad \text{supp}_2(\pi \odot_1 x) = \text{supp}_2(x).$$

**Proof** We only need to show one case, the other one following analogously. Given

two atoms  $a$  and  $a'$ , we have

$$(a a') \odot_1 \pi \odot_2 x = \pi \odot_2 (a a') \odot_1 x.$$

Since renaming operations are injective, we see that  $(a a') \odot_1 x = x$  if and only if  $(a a') \odot_1 \pi \odot_2 x = \pi \odot_2 x$ . But  $a$  is in  $\text{supp}_1(x)$  if and only if there are infinitely many  $a'$  such that  $(a a') \odot_1 x \neq x$ , and similarly for  $\pi \odot_2 x$ . This allows us to conclude.  $\square$

**Lemma 6.3** *Let  $X$  be a set with two independent nominal structures. We can define a compound nominal structure on  $X$  by setting*

$$\pi \cdot x \triangleq \pi \odot_1 \pi \odot_2 x.$$

*Each of the  $\odot_i$  is equivariant with respect to this compound operation, in the following sense:*

$$\pi \cdot \pi' \odot_i x = (\pi \cdot \pi') \odot_i \pi \cdot x.$$

*Finally, the support of an element is the union of the supports of the constituent parts:*

$$\text{supp}(x) = \text{supp}_1(x) \cup \text{supp}_2(x).$$

**Proof** By unpacking definitions, we can check directly that this compound operation indeed satisfies the required properties of a renaming operation, and that each  $\odot_i$  is equivariant. We can also see that every element is finitely supported: given  $x$  in  $X$  and a permutation  $\pi$  such that  $\pi \# \text{supp}_1(x) \cup \text{supp}_2(x)$ , we have

$$\pi \cdot x = \pi \odot_1 \pi \odot_2 x = \pi \odot_1 x = x,$$

proving that  $\text{supp}(x)$  exists and that it is contained in  $\text{supp}_1(x) \cup \text{supp}_2(x)$ . Note that  $\text{supp}_1(x) \cup \text{supp}_2(x)$  depends equivariantly on  $x$ , thanks to Lemma 6.2:

$$\begin{aligned} & \text{supp}_1(\pi \cdot x) \cup \text{supp}_2(\pi \cdot x) \\ &= \text{supp}_1(\pi \odot_1 \pi \odot_2 x) \cup \text{supp}_2(\pi \odot_2 \pi \odot_1 x) \\ &= \pi \cdot \text{supp}_1(\pi \odot_2 x) \cup \pi \cdot \text{supp}_2(\pi \odot_1 x) \\ &= \pi \cdot \text{supp}_1(x) \cup \pi \cdot \text{supp}_2(x) \\ &= \pi \cdot (\text{supp}_1(x) \cup \text{supp}_2(x)). \end{aligned}$$

Thus,  $\text{supp}_1(x) \cup \text{supp}_2(x)$  is also contained in  $\text{supp}(x)$ , which proves that both sets are equal.  $\square$

By iterating this process, we can combine any finite number of independent renaming operations. For simplicity, we restrict ourselves to the case of two independent operations in what follows, but the theory developed here can be generalized without difficulty to the case of a finite number of renaming operations that are pairwise independent. Whenever a set has multiple nominal structures, we consider the compound one defined in the above lemma as canonical.

## 6.2 Local Equivariance and Binding Operators

If a nominal set can be decomposed into independent renaming operations, we can express the scope of a binder on that set by instantiating the generic notion of

$\alpha$ -equivalence in Definition 3.1 to a particular renaming operation. However, if we want the corresponding quotient to behave nicely with respect to the “global” nominal structure, we must require that the corresponding binding operator be independent of the other renaming operations. This leads to *local* variants of the notions of equivariance and binding operator.

**Definition 6.4** Let  $X$  be a set with two independent nominal structures, and  $Y$  be a nominal set. We say that a function  $f$  is *locally equivariant* (with respect to  $\odot_i$ ) if

$$f(\pi \odot_j x) = \begin{cases} \pi \cdot f(x) & \text{if } j = i \\ f(x) & \text{otherwise} \end{cases}.$$

We note the set of such functions as  $X \rightarrow_{\text{eq}}^i Y$ .

By Lemma 6.2, we see that  $\text{supp}_i$  is locally equivariant with respect to the corresponding nominal structure. Furthermore:

**Lemma 6.5** *Using the same notations as above, a function  $f : X \rightarrow_{\text{eq}}^i Y$  is also equivariant with respect to the compound nominal structure of Lemma 6.3.*

**Proof** Assuming  $i = 1$ , we have  $f(\pi \cdot x) = f(\pi \odot_1 \pi \odot_2 x) = \pi \cdot f(\pi \odot_2 x) = \pi \cdot f(x)$ . The other case is similar.  $\square$

**Definition 6.6** Let  $X$  be a set with two independent nominal structures. A *local binding operator* (with respect to  $\odot_i$ ) is a locally equivariant function  $l : X \rightarrow_{\text{eq}}^i \mathcal{P}_{\text{fin}}(\mathbb{A})$ .

By Lemma 6.5, a local binding operator  $l$  for a renaming operation  $\odot_i$  is a binding operator for two different nominal structures, and thus gives rise to two different notions of  $\alpha$ -equivalence. To distinguish between them, we use  $x_1 \equiv_l x_2$  to say that  $x_1$  and  $x_2$  are  $\alpha$ -equivalent with respect to the compound structure, and  $x_1 \equiv_l^i x_2$  to say that  $x_1$  and  $x_2$  are  $\alpha$ -equivalent with respect to  $\odot_i$ . If we unfold the definition of  $\alpha$ -equivalence for the last case, it simply means that there exists a permutation  $\pi$  fixing  $\text{supp}_i(x) \setminus l(x)$  such that  $x_2 = \pi \odot_i x_1$ . Its corresponding quotient is compatible with all the nominal structures of the original set, as shown below.

**Lemma 6.7** *Let  $l : X \rightarrow_{\text{eq}}^i \mathcal{P}_{\text{fin}}(\mathbb{A})$  be a local binding operator. The relation  $\equiv_l^i$  is equivariant with respect to the operations  $\odot_j$  and with respect to  $\cdot$ . The quotient  $X/\equiv_l^i$  has the following nominal structures:*

$$\begin{aligned} \pi \odot_j [x] &= [\pi \odot_j x] & \text{supp}_j([x]) &= \begin{cases} \text{supp}_i(x) \setminus l(x) & \text{if } j = i \\ \text{supp}_j(x) & \text{otherwise} \end{cases} \\ \pi \cdot [x] &= [\pi \cdot x] & \text{supp}([x]) &= \text{supp}_i(x) \setminus l(x) \cup \text{supp}_{i'}(x), \end{aligned}$$

where  $i' \neq i$  in the last equation. In particular, the renaming operations  $\odot_j$  are independent.

**Proof** We assume  $i = 1$ , the other case being symmetric. It suffices to show the result for  $\odot_1$  and  $\odot_2$ , since these two cases combined yield the results for  $\cdot$ . Let's start with equivariance. We already know that  $\equiv_l^1$  is equivariant with respect to  $\odot_1$  from Lemma 3.3. To show that it is also the case for  $\odot_2$ , suppose that we have a permutation  $\pi$  such that  $\pi \# \text{supp}_1(x) \setminus l(x)$ , so that  $x \equiv_l^1 \pi \odot_1 x$ . We want to show that, for any permutation  $\pi'$ , we have

$$\pi' \odot_2 x \equiv_l^1 \pi' \odot_2 \pi \odot_1 x = \pi \odot_1 \pi' \odot_2 x.$$

This holds because, by local equivariance,  $\pi$  is fresh for  $\text{supp}_1(\pi' \odot_2 x) \setminus l(\pi' \odot_2 x) = \text{supp}_1(x) \setminus l(x)$ .

Finally, the definition of the renaming operations on  $X/\equiv_l^1$ , and their independence, follow from equivariance. We already know how to compute  $\text{supp}_1([x])$  from the earlier Lemma 3.4. Thus, to conclude, we just have to compute  $\text{supp}_2([x])$ . But  $x_1 \equiv_l^1 x_2$  implies  $\text{supp}_2(x_1) = \text{supp}_2(x_2)$  by Lemma 6.2, and thus  $\text{supp}_2([x]) = \text{supp}_2(x)$  (cf. (3) in the proof of Lemma 3.4).  $\square$

To understand how this construction works, let's revisit the example of parallel definitions of Section 4.4. Once again, if  $E$  is some nominal set of program expressions, we model raw parallel definitions (that is, before taking the quotient) with the nominal set  $(\mathbb{A} \rightarrow_{\text{fin}} E) \times E$ . We can decompose this nominal structure into two independent ones defined as

$$\pi \odot_1 (f, e) \triangleq (f \circ \pi^{-1}, \pi \cdot e) \qquad \pi \odot_2 (f, e) \triangleq (a \mapsto \pi \cdot f(a), e).$$

Thus, in a let expression

$$\text{let } a_1 = t_1 \text{ and } \dots \text{ and } a_n = t_n \text{ in } t,$$

the operation  $\odot_1$  renames the atoms on the left-hand side of the definitions, as well as the ones in  $t$ , whereas  $\odot_2$  only renames those that occur in the  $t_i$ . We see that the binding operator  $l(f, e) \triangleq \text{dom}(f)$  is local to  $\odot_1$ , and lists precisely the atoms on the left-hand side of the local definitions. Unlike the case of mutually recursive definitions discussed in Section 4.3, the definition of  $\equiv_l^1$  guarantees that the bound atoms of a pair  $(f, e)$  cannot vary in the bodies of local definitions in  $f$ . Thus, we can represent parallel let with the nominal set  $\text{Par}(E) = ((\mathbb{A} \rightarrow_{\text{fin}} E) \times E)/\equiv_l^1$ .

### 6.3 Elimination Principles

Now that we have quotient nominal sets that correspond to local binding operators, we turn our attention to the functions that can be defined on them. If we want a function that is locally equivariant with respect to the same nominal structure as the local binding operator that we considering, it suffices to apply Lemma 3.5 directly. More generally, however, we want to define functions that are *not* locally equivariant, but only equivariant with respect the compound nominal structure. Going back to the example of parallel let, the function  $c(f, e) = |\text{supp}(f, e)|$  that counts the number of variables in an expression is not locally equivariant, since renaming parts of an expression independently may change its result. For instance,

the expressions

$$\text{let } a_1 = a_1 \text{ in } a_1$$

and

$$\text{let } a_1 = a_2 \text{ in } a_1$$

have a different number of variables, but can be obtained from each other by a local renaming.

We cannot describe these functions using the compact elimination principle of Lemma 3.5, since, as stated earlier, projecting into such a quotient is not a binding function. This can be seen, for instance, in the identity  $\text{supp}([x]) = \text{supp}_1(x) \setminus l(x) \cup \text{supp}_2(x)$  of Lemma 6.7, which implies in particular that an atom  $a$  may appear in the support of  $[x]$  even if it occurs in  $l(x)$ . As it turns out, we can express the quotient by a local binding operator on  $X$  as a quotient by a “global” binding operator on a nominal subset of  $X$ , where bound atoms are prevented from aliasing the ones that remain free after the quotient by  $\alpha$ -equivalence. Specifically, we now prove that  $X/\equiv_l^1$  is isomorphic to the quotient  $X_{\#l}/l$ , where  $l : X \rightarrow_{\text{eq}}^1 \mathcal{P}_{\text{fin}}(\mathbb{A})$  is a local binding operator, and

$$X_{\#l} = \{x \in X \mid l(x) \# \text{supp}_2(x)\}. \quad (4)$$

(Note that  $X_{\#l}$  is a nominal subset of  $X$  for its compound nominal structure, but not for any of the  $\odot_i$ .) In particular, using Lemma 3.5, this allows us to define equivariant functions  $X/\equiv_l^1 \rightarrow_{\text{eq}} Y$  (with respect to the compound nominal structure of  $X/\equiv_l^1$ ) through equivariant functions  $f : X_{\#l} \rightarrow_{\text{eq}} Y$  that satisfy  $l(x) \# f(x)$  for any  $x$  in  $X_{\#l}$ . We begin with the following results, showing how to avoid conflicting with sets of “bad” atoms when choosing concrete values for the ones that are bound.

**Lemma 6.8** *Let  $X$  be a nominal set with a binding operator  $l$ . Given  $\bar{x} \in X/l$  and a finite set of atoms  $A$ , we can find  $x \in X$  such that  $[x] = \bar{x}$  and  $l(x) \# A$ .*

**Proof** Pick any representative  $x'$  of  $\bar{x}$ . We cannot choose  $x$  to be  $x'$  right away, since in principle the set  $l(x')$  may not be fresh with respect to  $A$ . We can, however, rename the conflicting atoms to fresh values.

Choose a set of atoms  $A'$  such that  $|A'| = |l(x') \cap A|$  and  $A' \# (x', A)$ . By a cardinality argument, we can construct a permutation  $\pi$  that sends  $l(x') \cap A$  to  $A'$  while leaving all other atoms fixed. By construction,  $\pi$  does not affect the free atoms of  $x'$ ; formally,  $\text{supp}(\pi) = l(x') \cap A \cup A'$ , hence  $\pi \# \text{supp}(x') \setminus l(x')$ . This implies that  $[\pi \cdot x'] = [x'] = \bar{x}$ . We then can choose  $x$  to be  $\pi \cdot x'$ , provided that we show that its atoms are completely fresh (that is,  $\pi \cdot l(x') \# A$ ). The result follows because the definition of  $\pi$  implies that  $\pi \cdot l(x') = A' \cup l(x') \setminus A$ , and both parts are disjoint from  $A$ .  $\square$

**Lemma 6.9** *Let  $X$  be a nominal set with a binding operator  $l$ , and  $A$  a finite set of atoms. Let  $x_1$  and  $x_2$  be two elements of  $X$  such that  $x_1 \equiv_l x_2$ ,  $l(x_1) \# A$ , and  $l(x_2) \# A$ . There exists a permutation  $\pi$  such that  $\pi \# A$ ,  $\pi \# \text{supp}(x_1) \setminus l(x_1)$ , and  $x_2 = \pi \cdot x_1$ .*

**Proof** By the definition of  $\equiv_l$ , we can find some permutation  $\pi'$  that is fresh for  $\text{supp}(x_1) \setminus l(x_1)$ , and such that  $x_2 = \pi' \cdot x_1$ . By basic properties of permutations,

there exists a permutation  $\pi$  such that

$$\begin{aligned} \pi(a) &= \pi'(a) && \text{if } a \in l(x_1) \\ \text{supp}(\pi) &\subseteq l(x_1) \cup l(x_2). \end{aligned}$$

The set  $l(x_1) \cup l(x_2)$  is disjoint from  $\text{supp}(x_1) \setminus l(x_1)$  and  $A$ , implying that  $\pi$  is fresh for  $\text{supp}(x_1) \setminus l(x_1)$  and  $A$ . In order to conclude, it suffices to show that  $\pi \cdot x_1 = \pi' \cdot x_1$ , which holds because  $\pi$  and  $\pi'$  agree on  $\text{supp}(x_1)$ .  $\square$

We can now explain how local binding operators yield binding functions.

**Lemma 6.10** *Let  $X$  be a set with two independent nominal structures, and  $l$  a local binding operator on  $X$  with respect to the operation  $\odot_1$ . There exists an isomorphism  $X/\equiv_l^1 \cong X_{\#l}/l$  making the following diagram commute:*

$$\begin{array}{ccc} X & \longrightarrow & X/\equiv_l^1 \\ \uparrow & & \parallel \\ X_{\#l} & \longrightarrow & X_{\#l}/l \end{array}$$

In particular, to build an equivariant function  $\bar{f} : X/\equiv_l^1 \rightarrow_{\text{eq}} Y$ , it suffices to find an equivariant  $f : X_{\#l} \rightarrow_{\text{eq}} Y$  such that  $l(x) \# f(x)$  for every  $x$ ; then,  $\bar{f}([x]) = f(x)$  whenever  $x$  is in  $X_{\#l}$ .

**Proof** Consider the canonical projection into  $X/\equiv_l^1$  restricted to the nominal subset  $X_{\#l}$ . Call this function  $f$ . By Lemma 5.2, it suffices to show that  $f$  is binding, and that its corresponding binding operator,  $l_f$ , is equal to  $l$ . The last point follows by unfolding definitions and making use of the freshness constraints on the elements of  $X_{\#l}$ . Moreover, we can show that  $f$  is surjective using Lemma 6.8. The only part that is missing is showing that  $f(x_1) = f(x_2)$  implies  $x_1 \equiv_l x_2$  for any  $x_1$  and  $x_2$  in  $X_{\#l}$ . Note that  $f(x_1) = f(x_2)$  is equivalent to  $x_1 \equiv_l^1 x_2$ . Using Lemma 6.9, we find a permutation  $\pi$  that is fresh for  $\text{supp}_2(x_1)$  and  $\text{supp}_1(x_1) \setminus l(x_1)$  such that  $x_2 = \pi \odot_1 x_1$ . We must then show that  $x \equiv_l \pi \odot_1 x$ . The assumptions on  $\pi$  imply that

$$x = \pi \odot_2 x \tag{5}$$

$$\pi \# \text{supp}_1(x) \setminus l(x) \cup \text{supp}_2(x). \tag{6}$$

Thus, showing  $x \equiv_l \pi \odot_1 x$  is tantamount to showing  $x \equiv_l \pi \cdot x = \pi \odot_1 \pi \odot_2 x$ . We conclude using (6), which, given that  $l(x) \# \text{supp}_2(x)$ , is equivalent to  $\pi \# \text{supp}(x) \setminus l(x)$ ,  $\square$

By applying this result to the nominal set  $\text{Par}(E)$ , and unfolding definitions, we find that it is isomorphic to

$$\{(f, e) \in (\mathbb{A} \rightarrow_{\text{fn}} E) \times E \mid \text{dom}(f) \# \text{im}(f)\} / (\text{dom} \circ p_1),$$

where  $p_1$  is the first projection. This shows that even if we can construct elements of  $\text{Par}(E)$  by giving a let expression where some of the free atoms in the bodies of the local definitions are shadowed, when defining functions on that set, we can

assume that the locally defined atoms are disjoint from the ones that are free. We can see this fact as a restatement, in nominal terms, of Barendregt's well-known variable convention, of which Lemma 6.10 is a formal justification.

#### 6.4 Multiple Quotients

Although the above results have been stated for quotients by a single local binding operator, they can also be composed to derive elimination principles for quotients by multiple operators. For this, we can make use of the following simple observation, which allows us to combine the freshness constraints arising from multiple quotients.

**Lemma 6.11** *Let  $X$  be a nominal set with a binding operator  $l$ . Every nominal subset  $\bar{X}$  of  $X/l$  is of the form*

$$\{x \in X \mid [x] \in \bar{X}\}/l.$$

**Proof** By Lemma 5.2. □

As an example of elimination principle for multiple quotients, we have the following result.

**Lemma 6.12** *Let  $X$  be a set with two independent nominal structures and two local binding operators,  $l_1 : X \rightarrow_{\text{eq}}^1 \mathcal{P}_{\text{fin}}(\mathbb{A})$  and  $l_2 : X \rightarrow_{\text{eq}}^2 \mathcal{P}_{\text{fin}}(\mathbb{A})$ . Let  $\equiv_{l_1, l_2}$  be the composition of the equivalence relations  $\equiv_{l_1}^1$  and  $\equiv_{l_2}^2$ . There is an isomorphism  $X/\equiv_{l_1, l_2} \cong X_{\#l_1, l_2}/(l_1 \cup l_2)$  such that the following diagram commutes:*

$$\begin{array}{ccc} X & \longrightarrow & X/\equiv_{l_1, l_2} \\ \uparrow & & \parallel \\ X_{\#l_1, l_2} & \longrightarrow & X_{\#l_1, l_2}/(l_1 \cup l_2) \end{array}$$

where

$$X_{\#l_1, l_2} = \{x \in X \mid l_1(x) \# \text{supp}_2(x), l_2(x) \# \text{supp}_1(x)\}.$$

**Proof** Because both renaming operations are independent, the composition  $\equiv_{l_1, l_2}$  is indeed an (equivariant) equivalence relation. We can express the quotient by this equivalence relation as an iterated quotient, which, thanks to Lemma 6.10, has the form

$$X/\equiv_{l_1, l_2} \cong (X_{\#l_1}/l_1)_{\# \bar{l}_2}/\bar{l}_2,$$

where  $\bar{l}_2$  denotes the lifting of the binding operator  $l_2$  to  $X_{\#l_1}/l_1$ , using Lemma 3.5. By Lemma 6.11, we have

$$(X_{\#l_1}/l_1)_{\# \bar{l}_2} \cong \{x \in X_{\#l_1} \mid l_2(x) \# \text{supp}_1(x) \setminus l_1(x)\}/l_1 \quad (7)$$

Since  $l_2(x) \subseteq \text{supp}_2(x)$  for every  $x \in X$ , we can see that  $l_1(x) \# l_2(x)$  when  $x \in X_{\#l_1}$ . Thus, the right-hand side of (7) is precisely  $X_{\#l_1, l_2}/l_1$ . We conclude using the fact that

$$X_{\#l_1, l_2}/l_1/\bar{l}_2 \cong X_{\#l_1, l_2}/(l_1 \cup l_2),$$

thanks to Lemma 5.3. It is a tedious but straightforward exercise to check that the composition of these isomorphisms results in the above commuting diagram. □



It is worth spelling out explicitly a special case of this result.

**Lemma 6.13** *Let  $X, Y$  be nominal sets and  $l_X, l_Y$  be binding operators over them. Define the separated product to be the following nominal subset of  $X \times Y$ :*

$$(X, l_X) \otimes (Y, l_Y) \triangleq \{(x, y) \in X \times Y \mid x \# l_Y(y), l_X(x) \# y\}$$

*Let  $l(x, y) \triangleq l_X(x) \cup l_Y(y)$ . There is an isomorphism*

$$\sigma : X/l_X \times Y/l_Y \cong ((X, l_X) \otimes (Y, l_Y))/l$$

*satisfying  $\sigma([x], [y]) = [(x, y)]$  for all  $(x, y) \in (X, l_X) \otimes (Y, l_Y)$ .*

**Proof** As pointed out before, we can define two independent renaming structures on  $X \times Y$ :

$$\pi \odot_1 (x, y) = (\pi \cdot x, y) \qquad \pi \odot_2 (x, y) = (x, \pi \cdot y).$$

We can check that  $l_X$  and  $l_Y$  can be lifted to local binding operators  $l_1$  and  $l_2$  on  $X \times Y$ , and that the separated product  $(X, l_X) \otimes (Y, l_Y)$  is just  $(X \times Y)_{\#l_1, l_2}$ , as defined in Lemma 6.12. We conclude by noting that the product relation  $\equiv_{l_X} \times \equiv_{l_Y}$  is the composition of  $\equiv_{l_1}^1$  and  $\equiv_{l_2}^2$ , and that

$$X/l_X \times Y/l_Y \cong (X \times Y)/(\equiv_{l_X} \times \equiv_{l_Y}) \cong (X \times Y)_{\#l_1, l_2}/l,$$

where the last isomorphism follows from Lemma 6.12.  $\square$

With this result, we can finally state a strong elimination principle for binding operators, which describes finitely supported functions defined on their quotients.

**Lemma 6.14** *Let  $X$  and  $Y$  be nominal sets,  $l$  be a binding operator on  $X$ , and  $f : X \rightarrow_{\text{fs}} Y$  a finitely supported function that satisfies the following freshness condition for binders: if  $x$  is such that  $l_X(x) \# f$ , then  $l_X(x) \# f(x)$ . There exists  $\bar{f} : X/l_X \rightarrow_{\text{fs}} Y$  satisfying  $\bar{f}([x]) = f(x)$  for all  $x$  such that  $l_X(x) \# f$ .*

**Proof** Roughly, we can use the previous result to express  $\bar{f}$  as the partial application of a suitable evaluation function. Define the nominal set

$$F \triangleq \{g : X \rightarrow_{\text{fs}} Y \mid \forall x. l_X(x) \# g \Rightarrow l_X(x) \# g(x)\}.$$

Pose  $l_F(g) \triangleq \emptyset$  and  $l(g, x) \triangleq l_X(x)$ . Let  $P \triangleq (F, l_F) \otimes (X, l_X)$  and  $e : P \rightarrow_{\text{eq}} Y$  be the evaluation function  $e(g, x) \triangleq g(x)$ . By construction,  $e$  satisfies  $l(g, x) \# e(g, x)$  for every  $(g, x) \in P$ . Using Lemma 3.5, we can thus construct  $\bar{e} : P/l \rightarrow_{\text{eq}} Y$  such that  $\bar{e}([(g, x)]) = g(x)$ . We then pose  $\bar{f}(\bar{x}) \triangleq \bar{e}(\sigma([f], \bar{x}))$ , where  $\sigma$  is the isomorphism of Lemma 6.13.  $\square$

Unlike the case for equivariant functions, the mapping  $f \mapsto \bar{f}$  defined above is *not* bijective in general, because it only uses part of the information contained in its argument: in order to have  $\bar{f} = \bar{g}$ , we just have to guarantee that  $f(x) = g(x)$  for all  $x$  such that  $l_X(x) \# (f, g)$ .

Lemma 6.14 is the analog for binding operators of an earlier result on name abstractions [9], which says that we can obtain a function  $\bar{f} : [\mathbb{A}]X \rightarrow_{\text{fs}} Y$  by finding  $f : \mathbb{A} \times X \rightarrow_{\text{fs}} Y$  satisfying  $a \# f(a, x)$  when  $a \# f$ —exactly what we obtain by instantiating our result with the operator  $l_\alpha$  defining name abstraction.

## 7 Functorial Properties

So far, we have used binding operators to define individual syntactic constructs, but still have not determined when such constructs can be combined into valid complete grammars. Previous results show that this is possible in many cases, such as grammars given by *nominal signatures* [18]. This allows us for instance to define the set of  $\lambda$ -terms modulo  $\alpha$ -equivalence as the solution of the equation

$$\Lambda = \mathbb{A} + \Lambda^2 + [\mathbb{A}]\Lambda, \quad (8)$$

which says that a  $\lambda$ -term is either a variable, a pair of  $\lambda$ -terms representing an application, or the name abstraction of a  $\lambda$ -term, representing a function definition.

In this section, we extend these results to a large class of nominal sets defined with binding operators. It is standard to interpret definitions such as the one above as specifying the initial algebra of a certain functor. What makes the definition of  $\Lambda$  valid is that name abstraction can be made into a functor, and that this functor satisfies certain technical conditions needed for the construction of initial algebras. Thus, we want to determine which endofunctors on  $\mathbf{Nom}$  can be defined through binding operators and to study their properties. The first step is to recast some of the earlier definitions into a more structured form, showing that the process of quotienting by a binding operator is itself functorial.

**Definition 7.1** The category of binding operators  $\mathbf{Bnd}$  is defined as follows. Objects are pairs  $(X, l_X)$ , where  $X$  is a nominal set and  $l_X$  is a binding operator over  $X$ . When no ambiguity can arise, we use  $X$  to refer to the pair  $(X, l_X)$ , and we sometimes omit the  $X$  index from  $l_X$ . A morphism from  $X$  to  $Y$  is an equivariant function  $f : X \rightarrow_{\text{eq}} Y$  such that, for every  $x$  in  $X$ ,

$$l_Y(f(x)) = l_X(x) \cap \text{supp}(f(x)).$$

We note  $U : \mathbf{Bnd} \rightarrow \mathbf{Nom}$  the obvious forgetful functor that maps  $(X, l_X)$  to  $X$ .

Intuitively, this definition says that applying a morphism  $f$  in  $\mathbf{Bnd}$  to an argument  $x$  cannot change the status of the atoms in  $\text{supp}(x)$  from bound to free, or vice versa. Note, however, that applying  $f$  may still *remove* atoms from the support of  $x$  entirely, both bound and free. This restriction guarantees that every such  $f$  can be lifted to quotients, as shown in the following result.

**Lemma 7.2** *We can extend quotients by binding operators to a functor  $Q : \mathbf{Bnd} \rightarrow \mathbf{Nom}$  satisfying*

$$Q(X) = X/l_X \qquad Q(f)([x]) = [f(x)].$$

Note that the second identity says that the canonical projections form a natural transformation  $U \rightarrow Q$ . This functor has a right adjoint  $Z : \mathbf{Nom} \rightarrow \mathbf{Bnd}$ , which associates to a nominal set  $X$  the constant binding operator  $l(x) = \emptyset$ . Furthermore, the  $QZ$  is naturally isomorphic to the identity on  $\mathbf{Nom}$ , via the canonical projection into the quotient.

**Proof** We define the action of  $Q$  on morphisms by appealing to Lemma 3.5. Specifically, let  $X$  and  $Y$  be two objects in  $\mathbf{Bnd}$ , and  $f : X \rightarrow Y$  be a morphism between them. We know that  $f$  and  $[-]$  are equivariant, thus it suffices to show that  $l_X(x) \# [f(x)]$  for all  $x$  in  $X$ . Since  $\text{supp}([f(x)]) = \text{supp}(f(x)) \setminus l_Y(f(x))$ , this is equivalent to

$$l_X(x) \cap \text{supp}(f(x)) \setminus l_Y(f(x)) = \emptyset,$$

which readily follows from the fact that  $f$  is a morphism in  $\mathbf{Bnd}$ . It is easy to verify that this construction preserves identities and composition; thus,  $Q$  is indeed a functor.

To show that  $Z$  is right adjoint to  $Q$ , consider an equivariant function  $f : Q(X) \rightarrow_{\text{eq}} Y$ , where  $X \in \mathbf{Bnd}$  and  $Y$  is a nominal set. The composite  $g = f \circ [-]$  is an equivariant function that satisfies  $l_X(x) \# g(x)$  for every  $x$  in  $X$ . Thus,  $g$  is a morphism  $X \rightarrow Z(Y)$  in  $\mathbf{Bnd}$ . Conversely, given a morphism  $g : X \rightarrow Z(Y)$  in  $\mathbf{Bnd}$ , we can use Lemma 3.5 to factor it as  $f \circ [-]$ , with  $f : Q(X) \rightarrow Y$ . We can readily check that these constructions are mutually inverse, and natural in  $X$  and  $Y$ . The last assertion follows from Lemma 5.2.  $\square$

We note that the condition on morphisms of Definition 7.1 is not tight, in the sense that the above proof would still work with the weaker assumption

$$l_Y(f(x)) \supseteq l_X(x) \cap \text{supp}(f(x)),$$

which intuitively says that  $f$  may bind atoms that are free in  $x$ . The reason for choosing the stronger variant, as we will see, is that it allows us to characterize  $\mathbf{Bnd}$  as a category of coalgebras, which will play an important role later on, when studying functors involving quotients.

To define a functor on  $\mathbf{Nom}$  via binding operators, we can define a functor  $F : \mathbf{Nom} \rightarrow \mathbf{Bnd}$ , and then consider the composite  $QF$ . It is easy to see that the examples discussed so far—name abstractions, name restriction, mutually recursive and parallel definitions—can be extended into functors by following this recipe. For name abstractions, for instance, we can take  $F(X)$  to be  $\mathbb{A} \times X$ , endowed with the binding operator  $l_\alpha(a, x) = \{a\}$ , which can be extended to a functor in the obvious way.

### 7.1 Strengthening Quotients

Many functors derived from binding operators allow arbitrary finitely supported functions to be lifted, not just equivariant ones. A good example is name abstraction: given any finitely supported function  $f : X \rightarrow_{\text{fs}} Y$ , we can define  $[\mathbb{A}]f : [\mathbb{A}]X \rightarrow_{\text{fs}} [\mathbb{A}]Y$  satisfying  $[\mathbb{A}]f(\langle a \rangle x) = \langle a \rangle (f(x))$  whenever  $a \# f$ . In category-theory jargon, such functors are known as *enriched*.

Formally, to enrich a functor  $G : \mathbf{Nom} \rightarrow \mathbf{Nom}$  means to extend its action on morphisms to a family of equivariant functions  $(X \rightarrow_{\text{fs}} Y) \rightarrow_{\text{eq}} (G(X) \rightarrow_{\text{fs}} G(Y))$  satisfying the usual functor laws. An equivariant action is compatible with the structure of  $\mathbf{Nom}$ , which allows us to generalize properties of  $G$  to finitely supported functions. For instance, if  $G$  has an initial algebra, it supports a recursion scheme that for defining finitely supported functions.

If  $G$  is of the form  $QF$ , it can be enriched by appealing to the elimination principle of Lemma 6.14, but the quotient structure provides a more direct route. Indeed, it is well-known that enriching  $QF$  is equivalent to giving it a *strength*: a natural transformation  $\eta_{X,Y} : X \times QF(Y) \rightarrow QF(X \times Y)$  satisfying the laws depicted below.

$$\begin{array}{ccc}
 & (A \times B) \times QF(C) & \xrightarrow{\eta} QF((A \times B) \times C) \\
 & \downarrow & \downarrow \\
 1 \times QF(A) & \xrightarrow{\eta} QF(1 \times A) & \\
 & \downarrow & \\
 & QF(A) & \\
 & \downarrow A \times \eta & \\
 & A \times QF(B \times C) & \xrightarrow{\eta} QF(A \times (B \times C))
 \end{array}$$

Intuitively,  $\eta$  allows us to lift functions by currying the composite

$$(X \rightarrow_{\text{fs}} Y) \times QF(X) \xrightarrow{\eta} QF((X \rightarrow_{\text{fs}} Y) \times X) \xrightarrow{F(\epsilon)} QF(Y),$$

where  $\epsilon$  denotes the evaluation function  $(X \rightarrow_{\text{fs}} Y) \times X \rightarrow_{\text{eq}} Y$ . The strength laws then guarantee that the resulting action satisfies the required functor laws.

Now, note that the separated product  $\otimes$  of Lemma 6.13 admits a trivial extension into a bifunctor  $\mathbf{Bnd}^2 \rightarrow \mathbf{Bnd}$ , endowing  $\mathbf{Bnd}$  with the structure of a symmetric monoidal category, with unit  $Z(1)$ ; furthermore, its isomorphism  $\sigma : Q(X) \times Q(Y) \cong Q(X \otimes Y)$  is natural in  $X$  and  $Y$ , and satisfies all laws required to make  $Q$  a strong monoidal functor from  $(\mathbf{Bnd}, \otimes, Z(1))$  to  $(\mathbf{Nom}, \times, 1)$ . This allows us to strengthen  $QF$  by composition: it suffices to find a natural transformation  $\eta'_{X,Y} : Z(X) \otimes F(Y) \rightarrow F(X \times Y)$  in  $\mathbf{Bnd}$  satisfying laws analogous to the ones above. It is then a simple exercise to check that the composite

$$X \times QF(Y) \xrightarrow{\cong} QZ(X) \times QF(Y) \xrightarrow{\cong} Q(Z(X) \otimes F(Y)) \xrightarrow{Q\eta'} QF(X \times Y)$$

is a strength on  $QF$ . Indeed, all of the functors arising from binding operators studied here can be trivially strengthened in this fashion.

## 7.2 Preservation of Colimits and Initial Algebras

After analyzing the matter of strength, let's turn our attention to other properties of quotient functors—namely, which colimits they preserve. Among other things, this is useful for building initial algebras. The initial algebra of a functor  $G : \mathbf{Nom} \rightarrow \mathbf{Nom}$  is normally constructed as the colimit of the chain diagram

$$\emptyset \xrightarrow{\iota} G(\emptyset) \xrightarrow{G(\iota)} G^2(\emptyset) \xrightarrow{G^2(\iota)} \dots,$$

but this construction only makes sense if  $G$  preserves that colimit, which can often

be reduced to showing that the individual functors appearing in definition of  $G$  preserve colimits of the same shape.

Note that, since  $Q$  has a right adjoint,  $QF$  preserves all colimits that are preserved by  $F$ . But  $F$  takes values in a category of binding operators, which must in principle be taken into account when computing these colimits. We show here is that this is not the case: we can always reduce colimits in  $\mathbf{Bnd}$  to simpler colimits in  $\mathbf{Nom}$ , by characterizing the former as the Eilenberg-Moore category of the following comonad.

**Lemma 7.3** *The  $L$  construction used in Section 4.2 for modeling name restriction can be extended into a functor  $\mathbf{Nom} \rightarrow \mathbf{Nom}$  by setting*

$$L(f)(A, x) = (A \cap \text{supp}(f(x)), f(x))$$

*This functor has the structure of a comonad, given by natural transformations  $\rho : L \rightarrow 1$  and  $\nu : L \rightarrow L^2$  defined by*

$$\rho(A, x) = x \qquad \nu(A, x) = (A, (A, x))$$

*and satisfying the usual conditions:  $L\nu \circ \nu = \nu L \circ \nu$  and  $L\rho \circ \nu = \rho L \circ \nu = 1_L$ .*

**Proof** It is easy to check that the action of  $L$  on morphisms is functorial; for instance,

$$\begin{aligned} L(f)(L(g)(A, x)) &= (A \cap \text{supp}(g(x)) \cap \text{supp}(f(g(x))), f(g(x))) \\ &= (A \cap \text{supp}(f(g(x))), f(g(x))) \\ &= L(f \circ g)(x), \end{aligned}$$

where we made use of the fact that  $\text{supp}(f(g(x))) \subseteq \text{supp}(g(x))$ . Checking that  $(L, \rho, \nu)$  forms a comonad is similarly straightforward.  $\square$

**Theorem 7.4** *The category  $\mathbf{Bnd}$  is equivalent to the Eilenberg-Moore category of coalgebras of the comonad  $(L, \rho, \nu)$ . We recall that objects of this category are pairs  $(X, l)$  of a nominal set  $X$  and a map  $l : X \rightarrow_{\text{eq}} L(X)$  satisfying the first two laws depicted below. A morphism from  $(X, l_X)$  to  $(Y, l_Y)$  is an equivariant function  $f : X \rightarrow_{\text{eq}} Y$  making the third diagram below commute.*

$$\begin{array}{ccccc} X & \xrightarrow{l} & L(X) & & X & \xrightarrow{l} & L(X) & & X & \xrightarrow{f} & Y \\ & \searrow 1 & \downarrow \rho & & \downarrow l & & \downarrow \nu & & \downarrow l_X & & \downarrow l_Y \\ & & X & & L(X) & \xrightarrow{L(l)} & L^2(X) & & L(X) & \xrightarrow{L(f)} & L(Y) \end{array}$$

**Proof** An equivariant function  $l : X \rightarrow_{\text{eq}} L(X)$  satisfying the commuting triangle above is of the form  $l(x) = (l'(x), x)$ , proving that such a function is equivalent to a binding operator on  $X$ . The first commuting square is valid for any  $l$  satisfying the triangle. The second commuting square is just a restatement of the restriction imposed on morphisms in  $\mathbf{Bnd}$ .  $\square$

As with every Eilenberg-Moore category, we obtain a right adjoint  $\bar{L}$  to the forgetful functor  $U : \mathbf{Bnd} \rightarrow \mathbf{Nom}$ . By unpacking the definitions, we can see this

right adjoint as endowing each  $L(X)$  with a binding operator  $l(A, x) \triangleq A$ , exactly what we used to model name restriction in Section 4.2. We now have all the required ingredients to prove the main result of this section.

**Theorem 7.5** *Let  $\mathcal{I}$  be a small category and  $D : \mathcal{I} \rightarrow \mathbf{Bnd}$  a diagram. Suppose that  $(U(\rho_i) : U(D_i) \rightarrow U(C))_{i \in \mathcal{I}}$  is a colimiting cocone in  $\mathbf{Nom}$ . Then so is  $(Q(\rho_i) : Q(D_i) \rightarrow Q(C))_{i \in \mathcal{I}}$ . In particular, any functor of the form  $QF$  preserves all colimits that are preserved by  $UF$ .*

**Proof** As previously noted, since  $Q$  has a right adjoint, it suffices to show that  $(\rho_i : D_i \rightarrow C)$  is a colimiting cocone in  $\mathbf{Bnd}$ . But this holds because  $U$  creates colimits, thanks to general results on Eilenberg-Moore categories [2, Props. 4.1.4 and 4.3.1].  $\square$

**Remark 7.6** The preceding result does not hold for limits in general. For a counterexample, consider the functor  $S : \mathbf{Nom} \rightarrow \mathbf{Bnd}$ , defined as  $S(X) = (X, \text{supp})$ , with the obvious action on morphisms. We have (trivially) that  $US(\mathbb{A}^2) = US(\mathbb{A})^2 = \mathbb{A}^2$ . On the other hand,  $QS(\mathbb{A}^2)$  has two elements  $[(a, a)]$  and  $[(a, a')]$ , where  $a \neq a'$ , whereas the product  $QS(\mathbb{A})^2$  has only one.

As mentioned previously, one application of Theorem 7.5 is showing that a functor of the form  $QF$  can be used for defining grammars by initial algebras, which often follows from simple category-theoretic reasons. For instance, suppose that  $F = (X \times (-), l_\alpha)$  is the functor defining generalized name abstractions  $[X](-)$ , as in Section 4. Then  $UF = X \times (-)$ , which preserves all colimits because  $\mathbf{Nom}$  is cartesian closed.

Another potential application is providing sufficient conditions for the existence of right adjoints of quotient functors. One of the many corollaries of the adjoint functor theorem says that, for a functor  $\mathbf{Nom} \rightarrow \mathbf{Nom}$ , preserving arbitrary colimits and having a right adjoint are equivalent, because  $\mathbf{Nom}$  is a Grothendieck topos. If that functor is of the form  $QF$ , then Theorem 7.5 allows us to check only whether  $UF$  preserves arbitrary colimits. We immediately conclude, for the same reasons as before, that generalized name abstractions have a right adjoint. Although this particular right adjoint already had a good explicit characterization [3], we think that our construction helps shed light on the relation between binding and adjunctions.

## 8 Conclusion and Related Work

Binding operators are an expressive framework for defining binders for nominal sets, encompassing many constructs that have been previously proposed in the literature. Although it is not clear how much expressive power our operators add compared to previous approaches, we believe that they provide a uniform, concise explanation for many of the properties enjoyed by binding constructs, such as their elimination principles, functoriality, compatibility with inductive definitions, etc.

Since the early development of nominal sets, researchers have directed their attention to more general forms of binding than name abstraction. The simplest such construction is given by generalized name abstractions, studied by Gabbay [4] and others. Clouston [3] investigated some of their categorical properties, in particu-

lar the related notion of *separating function*, and adjunctions between generalized name abstractions and the so-called *freshening function space*. That work provides an explicit construction of this adjunction, which could be interesting to generalize to other quotients by binding operators.

The Nominal Isabelle package features a rich language for defining data types with binders [17], allowing users to specify which atoms are bound in values of a data type. Unlike the present work, their focus is not in offering a foundational definition of binding, but in providing a usable and flexible tool. One point of similarity is that they use a general class of functions to enumerate which atoms are bound. Although such functions are more limited than general binding operators, the mechanism is rich enough to capture interesting binders, including generalized name abstractions and free nominal restriction sets. One way in which Nominal Isabelle goes beyond our framework is by allowing two parts of a term to be renamed independently, and yet share the same set of bound atoms. For instance, assuming that we have two different atoms  $a$  and  $a'$ , this would allow to equate terms of the form

$$\text{Foo } \{a, a'\} (a, a') (a, a') = \text{Foo } \{a, a'\} (a, a') (a', a),$$

assuming that the definition of **Foo** is such that it binds the set  $\{a, a'\}$  separately on its second and third arguments; that is, we allow swapping the  $a$  and  $a'$  in the third argument independently of the second.

To our knowledge, the closest relative of binding operators and their quotients is the operation of *simple monoidal sum* studied by Schöpp [13, Section 3.3.2]; we quickly review that construction here, adapted to our conventions and notations. We start with an arbitrary equivariant function  $f : X \rightarrow_{\text{eq}} Y * A$ , where  $Y * A$  denotes the “full” separated product, in which both components are not allowed to share any atoms. We then define a binding operator  $l$  on  $X$  by posing  $l(x) = \text{supp}(p_2(f(x)))$ . By construction,  $l(x) \# p_1(f(x))$  for every  $x$ . Thus, we can lift  $g = p_1 \circ f$  to  $\bar{g} : X/l \rightarrow_{\text{eq}} Y$ , which we call the simple monoidal sum of  $f$ . Viewed this way, this construction is a small generalization of quotients by binding operators; indeed, we can recover the latter by taking  $A$  to be  $\mathcal{P}_{\text{fin}}(\mathbb{A})$ . The main difference between both works is that Schöpp uses simple monoidal sums to interpret a form of dependent sum in a nominal type theory, studying properties of that construction that are more relevant in that context, whereas we attempt to provide a more elementary presentation of binding, quotients, and their properties.

## References

- [1] F. Borceux. *Handbook of Categorical Algebra: Volume 1, Basic Category Theory*. Cambridge Textbooks in Linguistics. Cambridge University Press, 1994.
- [2] F. Borceux. *Handbook of Categorical Algebra: Volume 2, Categories and Structures*. Cambridge Studies in Philosophy. Cambridge University Press, 1994.
- [3] R. Clouston. Generalised name abstraction for nominal sets. In *FoSSaCS*, volume 7794 of *Lecture Notes in Computer Science*, pages 434–449. Springer, 2013.
- [4] M. J. Gabbay. FM-HOL, a higher-order theory of names. In *In Thirty Five years of Automath, Heriot-Watt*, 2002.
- [5] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13(3-5):341–363, 2002.

- [6] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, Jan. 1993.
- [7] A. Kurz, D. Petrian, P. Severi, and F. de Vries. Nominal coalgebraic data types with applications to lambda calculus. *Logical Methods in Computer Science*, 9(4), 2013.
- [8] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, Sept. 1992.
- [9] A. M. Pitts. Alpha-structural recursion and induction. *J. ACM*, 53(3):459–506, May 2006.
- [10] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, New York, NY, USA, 2013.
- [11] A. M. Pitts and M. J. Gabbay. A metalanguage for programming with bound names modulo renaming. In *Mathematics of Program Construction, volume 1837 of Lecture Notes in Computer Science*, pages 230–255. Springer-Verlag, 2000.
- [12] F. Pottier. An overview of Caml. *Electron. Notes Theor. Comput. Sci.*, 148(2):27–52, Mar. 2006.
- [13] U. Schpp. *Names and Binding in Type Theory*. PhD thesis, University of Edinburgh, 2006.
- [14] P. Sewell, F. Z. Nardelli, S. Owens, G. Peskine, T. Ridge, S. Sarkar, and R. Strnisa. Ott: Effective tool support for the working semanticist. *J. Funct. Program.*, 20(1):71–122, 2010.
- [15] M. R. Shinwell, A. M. Pitts, and M. J. Gabbay. FreshML: Programming with binders made simple. In *Proceedings of the 8th ACM SIGPLAN International Conference on Functional Programming (ICFP 2003)*, volume 38, page 263274. ACM Press, August 2003.
- [16] C. Urban. Nominal techniques in Isabelle/HOL. *J. Autom. Reason.*, 40(4):327–356, May 2008.
- [17] C. Urban and C. Kaliszyk. General bindings and alpha-equivalence in nominal Isabelle. *Logical Methods in Computer Science*, 8(2), 2012.
- [18] C. Urban, A. Pitts, and M. Gabbay. Nominal unification. In *Computer Science Logic: 17th International Workshop CSL 2003, 12th Annual Conference of the EACSL, 8th Kurt Gödel Colloquium, KGC 2003, Vienna, Austria, August 25-30, 2003. Proceedings*, pages 513–527, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.



# Iteration and labelled iteration

Bram Geron and Paul Blain Levy

---

## Abstract

We analyse the conventional sum-based representation of iteration from the perspective of programmers, and show that the syntax they suggest is fundamentally not a good representation of Java-style iteration with `for`, `while`, `break`, and `continue`. We present an alternative syntax, which we call “labelled iteration”, where loops are identified using labels. The languages are analysed: we give denotational and operational semantics, adequacy proofs for both languages, and a translation function from sum-based iteration to labelled iteration.

*Keywords:* iteration, loops, lexical binding, operational semantics, denotational semantics, higher-order language, lambda calculus, de Bruijn indices

---

## 1 Introduction

*There might be an improved version of this article online  
at <http://bram.xyz/iteration> .*

### 1.1 Overview

Iteration is an important programming language feature.

- In imperative languages, it is best known in `for` and `while` loops. The meaning of such a loop is to iterate code until some condition is met, or if the condition is never met, the loop diverges. Such loops are often supplemented by `break` and `continue`.
- It has also been studied in the lambda calculus setting [13,19,21].
- In the categorical setting, iteration corresponds to complete Elgot monads [9]. They descend from iterative, iteration, and Elgot theories, and their algebras and monads [7,1,2,3,23], which study variants of the sum-based iteration  $-^\dagger$ . This field is related to Kleene monads [10,17,18].

Iteration can be implemented using recursion, but it is simpler: semantics of recursion require a least fixpoint, where iteration has a simple set-based semantics. Also from the programmer’s perspective, iteration and recursion are different: a program using a `for` or `while` loop can sometimes be clearer than the same program using recursion.

*This paper is electronically published in  
Electronic Notes in Theoretical Computer Science  
URL: [www.elsevier.nl/locate/entcs](http://www.elsevier.nl/locate/entcs)*

### 1.2 The sum-based representation of iteration

We study two representations of iteration. First, the classical sum-based construct  $-^\dagger$  that turns a computation  $\Gamma, A \vdash M : A + B$  into a computation  $\Gamma, A \vdash M^\dagger : B$ . Categorically, this representation of iteration corresponds to complete Elgot monads [9]. To understand the correspondence better, we introduce a term constructor `iter` for  $-^\dagger$ . (Details are in Section 2.)

$$\frac{\Gamma \Vdash V : A \quad \Gamma, x:A \Vdash M : A + B}{\Gamma \Vdash \text{iter } V, x. M : B}$$

Imperative programs with `for` and `while` can now be encoded using `iter`. As an example, the program on the left corresponds to the term on the right:

imperative	$\lambda$ -calculus-like
<pre> x := V; while (p(x)) {   x := f(x); } return g(x); </pre>	<pre> iter V, x.   if p(x)   then return inl f(x)   else return inr g(x) </pre>

This works as follows. The `iter` construct introduces a new identifier  $x$ , which starts at  $V$ . The body is evaluated. If the body evaluates to `inl`  $W$ , then the loop is finished and its result is  $W$ . If the body evaluates to `inl`  $V'$ , then we set  $x$  to  $V'$ , and keep on evaluating the body until it evaluates to some `inl`  $W$ .

### 1.3 The “De Bruijn index” problem with the sum-based representation

Programmers using imperative languages regularly use nested loops, as well their associated `break` and `continue` statements, which may be labelled. Such statements are not essential for programming, and code using `break` or `continue` can be rewritten to not use either statement, but this usually comes at a price in readability. There is usually a labelled and an unlabelled form of `break` and `continue`; here is an example imperative

program with nested, labelled, loops and labelled `continue` statements.

```

outer:    while (...) {
middle:   while (...) {
            if (...) {  $y := f(y)$ ; continue middle; }
            else
inner:    while (...) {
            if (...) {  $z := g(z)$ ; continue inner; }
            if (...) {  $y := h(y)$ ; continue middle; }
            if (...) {  $x := k(x)$ ; continue outer; }
            :
            }
        }
    }

```

Recall that “continue middle;” aborts the inner loop, as well as this iteration of the middle loop, and continues with a fresh iteration of the middle while loop. Statements “continue inner;” and “continue outer;” work analogously.

Assume that the outer, middle, and inner loops compute  $x$ ,  $y$ , and  $z$ , respectively. Then this program approximately corresponds to a term of the following structure.

```

iter  $V_{\text{init},x}, x$ .
  iter  $V_{\text{init},y}, y$ .
    if ... then return inl  $f(y)$ 
    else
      iter  $V_{\text{init},z}, z$ .
        if ... then return inl  $g(z)$ 
        else if ... then return inr inl  $h(y)$ 
        else if ... then return inr inr inl  $k(x)$ 
        else ...

```

- We encode the sequence “ $z := g(z)$ ; continue inner;” as “return inl  $g(z)$ ”, as in the first example; this will restart the inner iter with a new value for  $z$ .
- We encode the sequence “ $y := h(y)$ ; continue middle;” as “return inr inl  $h(y)$ ”: the body of the inner iter evaluates to inr inl  $h(y)$ , then the inner iter evaluates to inl  $h(y)$ , then the middle iteration is restarted with a new value for  $y$ .

In general, we can encode “ $w := m(w)$ ; continue label;” as

$$\text{return } \underbrace{\text{inr inr } \dots \text{ inr inl}}_{n \text{ times}} m(w)$$

where  $n$  is the number of loops that must be exited totally.

Such a representation of labelled `continue` is inadequate for programmers for three reasons.

- (i) To determine what loop the control statement refers to, the reader must resort to counting, which is error-prone.
- (ii) The sum-based representation of the same control statement looks different depending on the number of loops that must be skipped. That is, the representation of “continue outer” looks different depending on whether it occurs in the inner loop, the middle loop, or the outer loop.
- (iii) The same representation of a control statement, occurring in different places, can refer to a different loop. In our example, `return inl` corresponds to both `continue middle`; and to `continue inner`; .

We call this the “De Bruijn index problem”, because De Bruijn’s indices [6] for identifiers in  $\lambda$  calculus also work by counting intermediate binders, and they share these three problems for programmers. Indeed, De Bruijn [6] claims his notation to be good for a number of things, but does not claim that it is “easy to write and easy to read for the human reader”. For a brief introduction to De Bruijn indices, we refer to [14]; the same problem has also been studied from a different angle in [4,22].

#### 1.4 The solution: Labelled iteration

We solve the De Bruijn index problem with a second iteration construct, which we call labelled iteration. It binds a name  $x : A$  with a dual purpose: (1) It holds a *value* of type  $A$ . (2) It serves as a label for *restarting the loop*, upon which a new value of type  $A$  must be supplied.

For instance, the last example would look as follows using labelled iteration:

```

iter  $V_{\text{init},x}, x.$ 
  iter  $V_{\text{init},y}, y.$ 
    if  $\dots$  then raise $y$   $f(y)$ 
    else
      iter  $V_{\text{init},z}, z.$ 
        if  $\dots$  then raise $z$   $g(z)$ 
        else if  $\dots$  then raise $y$   $h(y)$ 
        else if  $\dots$  then raise $x$   $k(x)$ 
        else  $\dots$ 

```

Like sum-based iteration, labelled iteration has a set-based semantics, but the type system is more involved. We explain labelled iteration in more detail in Section 3. We chose the spelling *raise* because there is a similarity with raising an exception; see also the discussion in Section 4.

#### 1.5 Contributions

We define both languages: we give a type system, denotational semantics, big-step operational semantics, and an adequacy theorem for both languages. We explain the De Bruijn

values	$V, W ::= x \mid \langle \rangle \mid 0 \mid \text{succ } V \mid \text{inl } V \mid \text{inr } V \mid \langle V, W \rangle \mid \lambda x. M$
computations	$M, N ::= \text{return } V \mid \text{let } V \text{ be } x. M \mid M \text{ to } x. N$ $\mid V W \mid \text{case } V \text{ of } \{0. M; \text{succ } x. N\}$ $\mid \text{case } V \text{ of } \{\text{inl } x. M; \text{inr } y. N\} \mid \text{case } V \text{ of } \langle x, y \rangle. M$
types	$A, B, C ::= 1 \mid \text{nat} \mid A + B \mid A \times B \mid A \rightarrow B$

$\frac{(x : A) \in \Gamma}{\Gamma \vDash x : A}$	$\frac{}{\Gamma \vDash \langle \rangle : 1}$	$\frac{}{\Gamma \vDash 0 : \text{nat}}$	$\frac{\Gamma \vDash V : \text{nat}}{\Gamma \vDash \text{succ } V : \text{nat}}$
$\frac{\Gamma \vDash V : A}{\Gamma \vDash \text{inl } V : A + B}$	$\frac{\Gamma \vDash V : B}{\Gamma \vDash \text{inr } V : A + B}$	$\frac{\Gamma \vDash V : A \quad \Gamma \vDash W : B}{\Gamma \vDash \langle V, W \rangle : A \times B}$	
$\frac{\Gamma \vDash V : A}{\Gamma \vDash \text{return } V : A}$	$\frac{\Gamma \vDash V : A \quad \Gamma, x : A \vDash M : B}{\Gamma \vDash \text{let } V \text{ be } x. M : B}$	$\frac{\Gamma \vDash M : A \quad \Gamma, x : A \vDash N : B}{\Gamma \vDash M \text{ to } x. N : B}$	
$\frac{\Gamma, x : A \vDash M : B}{\Gamma \vDash \lambda x. M : A \rightarrow B}$	$\frac{\Gamma \vDash V : A \rightarrow B \quad \Gamma \vDash W : A}{\Gamma \vDash V W : B}$		
$\frac{\frac{\Gamma \vDash V : \text{nat} \quad \Gamma \vDash M : C \quad \Gamma, x : \text{nat} \vDash N : C}{\Gamma \vDash \text{case } V \text{ of } \{0. M; \text{succ } x. N\} : C} \quad \Gamma \vDash V : A + B \quad \Gamma, x : A \vDash M : C \quad \Gamma, y : B \vDash N : C}{\Gamma \vDash \text{case } V \text{ of } \{\text{inl } x. M; \text{inr } y. N\} : C}$ $\frac{\Gamma \vDash V : A \times B \quad \Gamma, x : A, y : B \vDash M : C}{\Gamma \vDash \text{case } V \text{ of } \langle x, y \rangle. M : C}$			

### Addition for sum-based iteration

$$\frac{\Gamma \vDash V : A \quad \Gamma, x : A \vDash M : A + B}{\Gamma \vDash \text{iter } V, x. M : B}$$

Figure 1. Above: syntax of plain fine-grain call-by-value. Sum-based iteration adds only one term construct and no values or types; the type derivation of this term is given below.

problem with the first language, and give a realistic example. We show that the first construct can be macro-expressed in terms of the second construct.

For both types of iteration, we study only loops with `continue`: we omit `break` because we believe it is a straightforward extension.

We define the language with sum-based iteration in Section 2, and the language with labelled iteration in Section 3.

## 2 Sum-based iteration

We define both our constructs in terms of fine-grain call-by-value or FGCBV [20], which

**Fine-grain call-by-value**

$$\begin{aligned}
\llbracket x \rrbracket_\rho &= \rho(x) & \llbracket \langle V, W \rangle \rrbracket_\rho &= \langle \llbracket V \rrbracket_\rho, \llbracket W \rrbracket_\rho \rangle \\
\llbracket \langle \rangle \rrbracket_\rho &= \langle \rangle & \llbracket \lambda x. M \rrbracket_\rho &= \lambda(a \in \llbracket A \rrbracket). \llbracket M \rrbracket_{(\rho, x \mapsto a)} \\
\llbracket 0 \rrbracket_\rho &= 0 & \llbracket \text{return } V \rrbracket_\rho &= \text{inl } \llbracket V \rrbracket_\rho \\
\llbracket \text{succ } V \rrbracket_\rho &= 1 + \llbracket V \rrbracket_\rho & \llbracket \text{let } V \text{ be } x. M \rrbracket_\rho &= \llbracket M \rrbracket_{(\rho, x \mapsto \llbracket V \rrbracket_\rho)} \\
\llbracket \text{inl } V \rrbracket_\rho &= \text{inl } \llbracket V \rrbracket_\rho & \llbracket M \text{ to } x. N \rrbracket_\rho &= \begin{cases} \llbracket N \rrbracket_{(\rho, x \mapsto v)} & \text{if } \llbracket M \rrbracket_\rho = \text{inl } v \\ \text{inr } \perp & \text{if } \llbracket M \rrbracket_\rho = \text{inr } \perp \end{cases} \\
\llbracket \text{inr } V \rrbracket_\rho &= \text{inr } \llbracket V \rrbracket_\rho & \llbracket V W \rrbracket_\rho &= \llbracket V \rrbracket_\rho \llbracket W \rrbracket_\rho
\end{aligned}$$

$$\begin{aligned}
\llbracket \text{case } V \text{ of } \{0. M; \text{succ } x. N\} \rrbracket_\rho &= \begin{cases} \llbracket M \rrbracket_\rho & \text{if } \llbracket V \rrbracket_\rho = 0 \\ \llbracket N \rrbracket_{(\rho, x \mapsto n)} & \text{if } \llbracket V \rrbracket_\rho = 1 + n \end{cases} \\
\llbracket \text{case } V \text{ of } \{\text{inl } x. M; \text{inr } y. N\} \rrbracket_\rho &= \begin{cases} \llbracket M \rrbracket_{(\rho, x \mapsto a)} & \text{if } \llbracket V \rrbracket_\rho = \text{inl } a \\ \llbracket N \rrbracket_{(\rho, y \mapsto b)} & \text{if } \llbracket V \rrbracket_\rho = \text{inr } b \end{cases} \\
\llbracket \text{case } V \text{ of } \{\langle x, y \rangle. M\} \rrbracket_\rho &= \llbracket M \rrbracket_{(\rho, x \mapsto a, y \mapsto b)} \quad \text{if } \llbracket V \rrbracket_\rho = \langle a, b \rangle
\end{aligned}$$

**Addition for sum-based iteration**

$$\llbracket \text{iter } V, x. M \rrbracket_\rho = \begin{cases} \text{inl } w & \text{if } \exists v_{0..k} \text{ s.t. } v_0 = \llbracket V \rrbracket_\rho \\ & \wedge \forall i : \llbracket M \rrbracket_{(\rho, x \mapsto v_i)} = \text{inl } \text{inl } v_{i+1} \\ & \wedge \llbracket M \rrbracket_{(\rho, x \mapsto v_k)} = \text{inl } \text{inr } w \\ \text{inr } \perp & \text{if no such } v_{0..k} \text{ exists} \end{cases}$$

Figure 2. Denotational semantics of values and terms in fine-grain call-by-value, and semantics of the sum-based iteration construct.

is a variant of call-by-value lambda calculus that has a syntactic separation between values and computations, and in which the evaluation order is made explicit.

We explain FGCBV and sum-based iteration here. The syntax and type system of FGCBV is given in Figure 1. We give a simple set-based semantics with divergence:

$$\begin{aligned}
\llbracket 1 \rrbracket &= \{\star\} & \llbracket \Gamma \rrbracket &= \prod_{(x:A) \in \Gamma} \llbracket A \rrbracket \\
\llbracket \text{nat} \rrbracket &= \mathbb{N} \\
\llbracket A + B \rrbracket &= \llbracket A \rrbracket + \llbracket B \rrbracket & \llbracket \Gamma \vdash^v V : A \rrbracket \in \llbracket \Gamma \rrbracket &\rightarrow \llbracket A \rrbracket \\
\llbracket A \times B \rrbracket &= \llbracket A \rrbracket \times \llbracket B \rrbracket & \llbracket \Gamma \vdash^e M : A \rrbracket \in \llbracket \Gamma \rrbracket &\rightarrow (\llbracket A \rrbracket + \{\perp\}) \\
\llbracket A \rightarrow B \rrbracket &= \llbracket A \rrbracket \rightarrow (\llbracket B \rrbracket + \{\perp\})
\end{aligned}$$

The semantics of plain FGCBV and FGCBV with sum-based iteration are the same, except of course that the latter has an extra construct. We give big-step operational semantics for both languages in Figure 3. The adequacy statements are simple:

**Proposition 2.1 (adequacy)**

- (i) For each closed term  $M$  of plain FGCBV without iteration, there is a unique  $V$  such that  $M \Downarrow \text{return } V$ , and  $\llbracket M \rrbracket_\emptyset = \text{inl } \llbracket V \rrbracket_\emptyset$ .
- (ii) For each closed term  $M$  of FGCBV with sum-based iteration, either

**Fine-grain call-by-value** $T ::= \text{return } V$ 

$$\begin{array}{c}
\frac{}{\text{return } V \Downarrow \text{return } V} \quad \frac{M[V/x] \Downarrow T}{\text{let } V \text{ be } x. M \Downarrow T} \quad \frac{M \Downarrow \text{return } V \quad N[V/x] \Downarrow T}{M \text{ to } x. N \Downarrow T} \\
\\
\frac{M[W/x] \Downarrow T}{(\lambda x. M) W \Downarrow T} \quad \frac{M[V/x, W/y] \Downarrow T}{\text{case } \langle V, W \rangle \text{ of } \{\langle x, y \rangle. M\} \Downarrow T} \\
\\
\frac{M_0 \Downarrow T}{\text{case } 0 \text{ of } \{0. M_0; \text{succ } x. M_{\text{succ}}\} \Downarrow T} \quad \frac{M_{\text{succ}}[V/x] \Downarrow T}{\text{case } (\text{succ } V) \text{ of } \{0. M_0; \text{succ } x. M_{\text{succ}}\} \Downarrow T} \\
\\
\frac{M_{\text{inl}}[V/x] \Downarrow T}{\text{case } (\text{inl } V) \text{ of } \{\text{inl } x. M_{\text{inl}}; \text{inr } x. M_{\text{inr}}\} \Downarrow T} \quad \frac{M_{\text{inr}}[V/x] \Downarrow T}{\text{case } (\text{inr } V) \text{ of } \{\text{inl } x. M_{\text{inl}}; \text{inr } x. M_{\text{inr}}\} \Downarrow T}
\end{array}$$

**Addition for sum-based iteration** $T ::= \text{return } V$ 

$$\frac{\exists k \geq 0 \exists (V_1, \dots, V_k) \forall i \in \{1..k\} : M[V_{i-1}/x] \Downarrow \text{return inl } V_i \quad M[V_k/x] \Downarrow \text{return inr } Z}{\text{iter } V_0, x. M \Downarrow \text{return } Z}$$

Figure 3. Big-step operational semantics of plain fine-grain call-by-value and of sum-based iteration. In our operational semantics, closed terms reduce to “terminal” terms of the same type, or they do not reduce at all. We use metavariable  $T$  for terminals. For FGCBV and its extension with sum-based iteration, terminal terms are always of the form  $\text{return } V$ . Introducing a separate notion of terminals might seem odd for now, but in Figure 6 we extend the rules for FGCBV and add another form of terminal. So  $T$  above may come to stand for something other than  $\text{return } V$  further in the paper.

- *there is a unique  $V$  such that  $M \Downarrow \text{return } V$ , and  $\llbracket M \rrbracket_{\emptyset} = \text{inl } \llbracket V \rrbracket_{\emptyset}$ , or*
- *$M$  does not reduce to a terminal, and  $\llbracket M \rrbracket_{\emptyset} = \text{inr } \perp$ .*

**3 Labelled iteration with pure function types***3.1 Introduction*

To fix the De Bruijn index problem indicated in Section 1.3, we now give a language that has an effectful “labelled iteration” construct instead. The judgements in this language are

$$\begin{array}{ll}
\Delta; \Gamma \vDash M : A & \text{for computations} \\
\Gamma \vDash V : A & \text{for values}
\end{array}$$

We give the typing rules in Figure 4.  $\Gamma$  is a context of identifiers representing values, as usual.  $\Delta$  exists only for computations; it is a context of *typed labels*. Denotations of

Values and types are the same as in fine-grain call-by-value in Figure 1 on page 199.

computations  $M, N ::= \dots \mid \text{iter } V, x. M \mid \text{raise}_x V$

$$\begin{array}{c}
\frac{(x:A) \in \Gamma}{\Gamma \Vdash x : A} \qquad \frac{\Gamma \Vdash V : A \quad \Delta; \Gamma, x:A \Vdash M : B}{\Delta; \Gamma \Vdash \text{let } V \text{ be } x. M : B} \\
\\
\frac{\Gamma \Vdash V : A}{\Delta; \Gamma \Vdash \text{return } V : A} \qquad \frac{\Delta; \Gamma \Vdash M : A \quad \Delta; \Gamma, x:A \Vdash N : B}{\Delta; \Gamma \Vdash M \text{ to } x. N : B} \\
\\
\frac{\cdot; \Gamma, x:A \Vdash M : B}{\Gamma \Vdash \lambda x. M : A \rightarrow B} \qquad \frac{\Gamma \Vdash V : A \rightarrow B \quad \Gamma \Vdash W : A}{\Delta; \Gamma \Vdash V W : B} \\
\\
\frac{}{\Gamma \Vdash \langle \rangle : 1} \qquad \frac{\Gamma \Vdash V : A}{\Gamma \Vdash \text{inl } V : A + B} \qquad \frac{\Gamma \Vdash V : B}{\Gamma \Vdash \text{inr } V : A + B} \\
\\
\frac{\Gamma \Vdash V : \text{nat} \quad \Delta; \Gamma \Vdash M : C \quad \Delta; \Gamma, x:A \Vdash N : C}{\Delta; \Gamma \Vdash \text{case } V \text{ of } \{0. M; \text{succ } x. N\} : C} \\
\frac{\Gamma \Vdash V : A + B \quad \Delta; \Gamma, x:A \Vdash M : C \quad \Delta; \Gamma, y:B \Vdash N : C}{\Delta; \Gamma \Vdash \text{case } V \text{ of } \{\text{inl } x. M; \text{inr } y. N\} : C} \\
\frac{\Gamma \Vdash V : A \times B \quad \Delta; \Gamma, x:A, y:B \Vdash M : C}{\Delta; \Gamma \Vdash \text{case } V \text{ of } \langle x, y \rangle. M : C} \\
\\
\frac{\Gamma \Vdash V : A \quad \Delta, x:A; \Gamma, x:A \Vdash M : B}{\Delta; \Gamma \Vdash \text{iter } V, x. M : B} \qquad \frac{\Gamma \Vdash V : A \quad (x:A) \in \Delta}{\Delta; \Gamma \Vdash \text{raise}_x V : B}
\end{array}$$

Figure 4. Syntax of labelled iteration.

judgements are

$$\begin{aligned}
\llbracket \Delta; \Gamma \Vdash A \rrbracket &= \left( \prod_{(x:B) \in \Gamma} \llbracket B \rrbracket \right) \rightarrow \left( \sum_{(x:B) \in \Delta} \llbracket B \rrbracket + \llbracket A \rrbracket + \{\perp\} \right) \\
\llbracket \Gamma \Vdash A \rrbracket &= \left( \prod_{(x:B) \in \Gamma} \llbracket B \rrbracket \right) \rightarrow \llbracket A \rrbracket .
\end{aligned}$$

$\Gamma$  is used to form values.

$$\frac{(x:A) \in \Gamma}{\Gamma \Vdash x : A}$$

$\Delta$  is used to form computations, much like raising an exception. However, conventionally, exception names come from a global set. Our “exception names”, which we call labels, will be bound lexically, much like identifiers are bound by  $\lambda$ .

Furthermore, when a label is raised, it must be parametrised by a value of the corresponding



type. The typing rule is as follows:

$$\frac{\Gamma \Vdash V : A \quad (x:A) \in \Delta}{\Delta; \Gamma \vdash \text{raise}_x V : B}$$

We thus have these judgements.

$$\begin{aligned} (x : \text{nat} \times \text{bool}) ; (y:\text{nat}, z:\text{bool}) &\vdash \text{raise}_x \langle 3, \text{true} \rangle : \text{string} \\ (x : \text{nat} \times \text{bool}) ; (y:\text{nat}, z:\text{bool}) &\vdash \text{raise}_x \langle y, z \rangle : 0 \\ (x : \text{nat} \times \text{bool}) ; (y:\text{nat}, z:\text{bool}) &\vdash \text{return } y : \text{nat} \end{aligned}$$

But we cannot raise identifiers:

$$(x : \text{nat} \times \text{bool}) ; (y:\text{nat}, z:\text{bool}) \not\vdash \text{raise}_y 3$$

And we can also not use labels for their value:

$$(x : \text{nat} \times \text{bool}) ; (y:\text{nat}, z:\text{bool}) \not\vdash \text{return } x : \text{nat} \times \text{bool}$$

Indeed, the typing rule of return (see Figure 4 on the previous page) shows that  $x$  is not available in the context of the argument to return:

$$\frac{y:\text{nat}, z:\text{bool} \Vdash V : \text{nat} \times \text{bool}}{(x : \text{nat} \times \text{bool}) ; (y:\text{nat}, z:\text{bool}) \vdash \text{return } V : \text{nat} \times \text{bool}}$$

Our use of a syntactically separate kind of names bears resemblance to the use of function names by Kennedy [16] for control.

### Labelled iteration

We now wish to use labels to generalise the iter  $V, x. M$  from last section. Remember that previously when  $M$  reduces to

- return inl  $V'$ ,                      then the loop should be re-tried with value  $V'$ ,
- return inr  $W$ ,                      then the result of the loop is  $W$ .

Our new notation will *also* be iter  $V, x. M$ . However, here  $x$  is *both* an identifier and a label:

$$\frac{\Gamma \Vdash V : A \quad \Delta, x:A; \Gamma, x:A \vdash M : B}{\Delta; \Gamma \vdash \text{iter } V, x. M : B}$$

Now similarly when writing iter  $V, x. M$ , when  $M$  reduces to

- $\text{raise}_x V'$ ,                      then the loop should be re-tried with value  $V'$ ,
- $\text{raise}_y W, (y \neq x)$               then the loop should be aborted
- and loop  $y$  should be re-tried with value  $W$ ,
- return  $W$ ,                      then the result of the loop is  $W$ .

We wish to repeat that the same name  $x$  can appear in *both*  $\Delta$  and  $\Gamma$ . We pose no general syntactic restriction on  $(x:A) \in \Delta$  and  $(x:B) \in \Gamma$  to have the same type. However, to be able to form  $\text{iter } V, x. M$ , we must have  $x$  in both  $\Delta$  and  $\Gamma$  of the same type.

We also wish to note at this point that we define the semantics of our language on the *binding diagrams*[8], that is, on the abstract syntax modulo  $\alpha$ -equivalence.

### Labelled iteration and $\lambda$

Now that contexts for computations are different from contexts for values, the conventional fine-grain call-by-value judgements have to be tweaked to work in this setting. The typing rule for `return` in Figure 4 is simple: when we move upwards from a computation to a value judgement we just forget about  $\Delta$ .

$$\frac{\Gamma \Vdash V : A}{\Delta; \Gamma \Vdash \text{return } V : A}$$

But reversely, for  $\lambda$ , we have a choice: what should  $\Delta$  be? We take what seems to be the only reasonable choice: to reset  $\Delta$  to the empty context,  $\cdot$ .

$$\frac{\cdot; \Gamma, x:A \Vdash M : B}{\Gamma \Vdash \lambda x. M : A \rightarrow B}$$

Java agrees with this choice: it is a syntax error to write a labelled `continue` or `break` with a label outside of the current method [11]. From a programmer's perspective, this means that all functions are pure.

### 3.2 Denotational semantics

Recall that the semantics of term and value judgements is as follows.

$$\begin{aligned} \llbracket \Delta; \Gamma \Vdash A \rrbracket &= \left( \prod_{(x:B) \in \Gamma} \llbracket B \rrbracket \right) \rightarrow \left( \sum_{(x:B) \in \Delta} \llbracket B \rrbracket + \llbracket A \rrbracket + \{\perp\} \right) \\ \llbracket \Gamma \Vdash A \rrbracket &= \left( \prod_{(x:B) \in \Gamma} \llbracket B \rrbracket \right) \rightarrow \llbracket A \rrbracket \end{aligned}$$

The denotation of types is as follows.

$$\begin{aligned} \llbracket 1 \rrbracket &= \{\star\} \\ \llbracket \text{nat} \rrbracket &= \mathbb{N} \\ \llbracket A + B \rrbracket &= \llbracket A \rrbracket + \llbracket B \rrbracket \\ \llbracket A \times B \rrbracket &= \llbracket A \rrbracket \times \llbracket B \rrbracket \\ \llbracket A \rightarrow B \rrbracket &= \llbracket A \rrbracket \rightarrow (\llbracket B \rrbracket + \{\perp\}) \end{aligned}$$

We give the semantics of terms and values in Figure 5. We use the following notation for elements of the ternary sum  $(\sum_{(x:B) \in \Delta} \llbracket B \rrbracket + \llbracket A \rrbracket + \{\perp\})$ :

- (i) *return*  $a$  (for  $a \in \llbracket A \rrbracket$ ) (compare to the term notation: `return V`),

$$\begin{array}{ll}
\llbracket x \rrbracket_\rho = \rho(x) & \llbracket \text{return } V \rrbracket_\rho = \text{return } \llbracket V \rrbracket_\rho \\
\llbracket \langle \rangle \rrbracket_\rho = \langle \rangle & \llbracket \text{raise}_x V \rrbracket_\rho = \text{raise}_x \llbracket V \rrbracket_\rho \\
\llbracket 0 \rrbracket_\rho = 0 & \llbracket \text{let } V \text{ be } x. M \rrbracket_\rho = \llbracket M \rrbracket_{(\rho, x \mapsto \llbracket V \rrbracket_\rho)} \\
\llbracket \text{succ } V \rrbracket_\rho = 1 + \llbracket V \rrbracket_\rho & \llbracket M \text{ to } x. N \rrbracket_\rho = \begin{cases} \llbracket N \rrbracket_{(\rho, x \mapsto v)} & \text{if } \llbracket M \rrbracket_\rho = \text{return } v \\ \text{raise}_y w & \text{if } \llbracket M \rrbracket_\rho = \text{raise}_y w \\ \perp & \text{if } \llbracket M \rrbracket_\rho = \perp \end{cases} \\
\llbracket \text{inl } V \rrbracket_\rho = \text{inl } \llbracket V \rrbracket_\rho & \llbracket V \ W \rrbracket_\rho = \llbracket V \rrbracket_\rho \llbracket W \rrbracket_\rho \\
\llbracket \text{inr } V \rrbracket_\rho = \text{inr } \llbracket V \rrbracket_\rho & \\
\llbracket \langle V, W \rangle \rrbracket_\rho = \langle \llbracket V \rrbracket_\rho, \llbracket W \rrbracket_\rho \rangle & \\
\llbracket \lambda x. M \rrbracket_\rho = \lambda(a \in \llbracket A \rrbracket). \llbracket M \rrbracket_{(\rho, x \mapsto a)} & \\
\\
\llbracket \text{case } V \text{ of } \{0. M; \text{succ } x. N\} \rrbracket_\rho = \begin{cases} \llbracket M \rrbracket_\rho & \text{if } \llbracket V \rrbracket_\rho = 0 \\ \llbracket N \rrbracket_{(\rho, x \mapsto n)} & \text{if } \llbracket V \rrbracket_\rho = 1 + n \end{cases} \\
\llbracket \text{case } V \text{ of } \{\text{inl } x. M; \text{inr } y. N\} \rrbracket_\rho = \begin{cases} \llbracket M \rrbracket_{(\rho, x \mapsto a)} & \text{if } \llbracket V \rrbracket_\rho = \text{inl } a \\ \llbracket N \rrbracket_{(\rho, y \mapsto b)} & \text{if } \llbracket V \rrbracket_\rho = \text{inr } b \end{cases} \\
\llbracket \text{case } V \text{ of } \{\langle x, y \rangle. M\} \rrbracket_\rho = \llbracket M \rrbracket_{(\rho, x \mapsto a, y \mapsto b)} & \text{if } \llbracket V \rrbracket_\rho = \langle a, b \rangle \\
\llbracket \text{iter } V, x. M \rrbracket_\rho = \begin{cases} \text{return } w & \text{if } \exists v_{0..k} \text{ s.t. } v_0 = \llbracket V \rrbracket_\rho \\ & \wedge \forall i : \llbracket M \rrbracket_{(\rho, x \mapsto v_i)} = \text{raise}_x v_{i+1} \\ & \wedge \llbracket M \rrbracket_{(\rho, x \mapsto v_k)} = \text{return } w \\ \text{raise}_y w & \text{if } \exists v_{0..k} \text{ s.t. } v_0 = \llbracket V \rrbracket_\rho \\ & \wedge \forall i : \llbracket M \rrbracket_{(\rho, x \mapsto v_i)} = \text{raise}_x v_{i+1} \\ & \wedge \llbracket M \rrbracket_{(\rho, x \mapsto v_k)} = \text{raise}_y w \\ \perp & \text{if no other case matches} \end{cases}
\end{array}$$

Figure 5. Denotational semantics of terms and values of the language with labelled iteration. See also Section 3.2.

(ii)  $\text{raise}_x b$  (for  $b \in \llbracket B \rrbracket$ ) (compare to the term notation:  $\text{raise}_x V$ ),

(iii)  $\perp$ .

**Definition 3.1** [weakening] We say that  $\Delta'; \Gamma'$  is *stronger* than  $\Delta; \Gamma$  when  $\Delta' \subseteq \Delta$  and  $\Gamma' \subseteq \Gamma$ . Alternatively, we say that  $\Delta; \Gamma$  is *weaker* than  $\Delta'; \Gamma'$ .

A term in a context is also a term in a weaker context, with the same derivation. A value in a context is also a value in a weaker context, with the same derivation.

**Definition 3.2** [closedness]

(i) When  $\cdot \Vdash V : A$ , then we say that  $V$  is *closed*.

(ii) When  $\Delta; \cdot \vDash M : A$ , then we say that  $M$  is *closed*.

**Definition 3.3** A *substitution* (between two-zone contexts)  $\sigma : \Delta'; \Gamma' \rightarrow \Delta; \Gamma$  consists of two parts,

- for every label  $(x : A) \in \Delta'$ , a label  $\sigma_{\text{lab}}(x)$  of type  $A$  in  $\Delta$ , and

$$\begin{array}{c}
T ::= \text{return } V \mid \text{raise}_x V \\
\\
\frac{M \Downarrow \text{raise}_x V}{M \text{ to } x. N \Downarrow \text{raise}_x V} \\
\\
\frac{\exists k \geq 0 \exists (V_1, \dots, V_k) \forall i \in \{1..k\} : M[V_{i-1}/x] \Downarrow \text{raise}_x V_i \quad M[V_k/x] \Downarrow \text{return } Z}{\text{iter } V_0, x. M \Downarrow \text{return } Z} \\
\\
\frac{\exists k \geq 0 \exists (V_1, \dots, V_k) \forall i \in \{1..k\} : M[V_{i-1}/x] \Downarrow \text{raise}_x V_i \quad M[V_k/x] \Downarrow \text{raise}_y Z}{\text{iter } V_0, x. M \Downarrow \text{raise}_y Z} (x \neq y)
\end{array}$$

Figure 6. Big-step operational semantics for labelled iteration. This figure extends Figure 3. Namely, we add rules, and we add a new form of terminal:  $\text{raise}_x V$ .

- for every identifier  $(x : A) \in \Gamma'$ , a *value*  $\sigma_{\text{id}}(x) \quad (\Gamma \Vdash \sigma_{\text{id}}(x) : A)$ .

**Remark 3.4** From a two-zone substitution  $\sigma : \Delta'; \Gamma' \rightarrow \Delta; \Gamma$  we can trivially obtain a one-zone substitution  $\Gamma' \rightarrow \Gamma$ . By abuse of notation, we also write  $\sigma$  for this obtained substitution on one-zone contexts. Similarly, from a one-zone substitution  $\sigma : \Gamma' \rightarrow \Gamma$ , we obtain trivially a two-zone substitution  $\cdot; \Gamma' \rightarrow \cdot; \Gamma$ , for which we also write  $\sigma$ .

We can use a substitution  $\sigma : \Delta'; \Gamma' \rightarrow \Delta; \Gamma$  as follows on terms. Given a term  $\Delta'; \Gamma' \Vdash M : A$ , we obtain the term  $\Delta; \Gamma \Vdash M\sigma : A$  by

- for any  $x \in \Delta$ , replacing all occurrences of  $\text{raise}_x V$  (where  $x$  is free) by  $\text{raise}_{\sigma_{\text{lab}}(x)} (V\sigma)$ , where  $V\sigma$  is given similarly by induction. And
- for any  $x \in \Gamma$ , replacing all value occurrences of identifiers by  $\sigma_{\text{id}}(x)$ .

For one-zone contexts  $\Gamma$  we have the usual notion of substitution  $\sigma : \Gamma' \rightarrow \Gamma$  that assigns a value (over  $\Gamma$ ) to each identifier of  $\Gamma'$ . And given  $\Gamma' \Vdash V : A$ , we obtain similarly  $\Gamma \Vdash V\sigma : A$ .

Two-zone contexts and their substitutions form a category, and one-zone contexts and their substitutions form another category. That is, substitutions can be composed associatively and composition has an identity.

**Lemma 3.5 (substitution lemma)**

- (i) Let one-zone substitution  $\sigma : \Gamma' \rightarrow \Gamma$  be given. If  $\Gamma' \Vdash V : A$ , then  $\llbracket V\sigma \rrbracket_\rho = \llbracket V \rrbracket_{(x \mapsto \llbracket \sigma(x) \rrbracket_\rho)_{x \in \Gamma'}}$ .
- (ii) Let two-zone substitution  $\sigma : \Delta'; \Gamma' \rightarrow \Delta; \Gamma$  be given. If  $\Delta'; \Gamma' \Vdash M : A$ , then  $\llbracket M\sigma \rrbracket_\rho = f(\llbracket M \rrbracket_{(x \mapsto \llbracket \sigma_{\text{id}}(x) \rrbracket_\rho)_{x \in \Gamma'}})$ , where  $f$  maps  $\text{raise}_x v$  to  $\text{raise}_{\sigma_{\text{lab}}(x)} v$ .

### 3.3 Operational semantics

We define a big-step “reduction” relation  $M \Downarrow T$  between closed terms  $\Delta; \cdot \Vdash M : A$  and (closed) terminals  $\Delta; \cdot \Vdash T : A$  of the same type, such that for every such  $M$  either

- (i)  $M \Downarrow T = \text{return } V$ , or

- (ii)  $M \Downarrow T = \text{raise}_x V$ ,  $x \in \text{dom } \Delta$ , or
- (iii)  $M$  does not reduce.

Derivation rules are given in Figure 6 on the preceding page, and the reduction relation is defined as their least fixed point.

**Theorem 3.6 (adequacy)**

- (i) If  $M \Downarrow \text{return } V$ , then  $\llbracket M \rrbracket_{\emptyset} = \text{return } \llbracket V \rrbracket_{\emptyset}$ .
- (ii) If  $M \Downarrow \text{raise}_x V$ , then  $\llbracket M \rrbracket_{\emptyset} = \text{raise}_x \llbracket V \rrbracket_{\emptyset}$ .
- (iii) If  $M$  does not reduce, then  $\llbracket M \rrbracket_{\emptyset} = \perp$ .

### 3.4 Translation from sum-based iteration

Let  $\Gamma \models M : A$  or  $\Gamma \models V : A$  be a term or value in the language with sum-based iteration. We define a *translation*  $\text{translate}(M)$ ,  $\text{translate}(V)$  from sum-based iteration, such that  $\cdot ; \Gamma \models \text{translate}(M) : A$  or  $\Gamma \models \text{translate}(V) : A$ , respectively, in the language with labelled iteration. The translation macro-expands sum-based iter as follows. The other constructs are left unchanged.

$$\begin{aligned} \text{translate}(\text{iter } V, x. M) = & \text{iter } V, x. (\text{translate}(M) \text{ to result.} \\ & \text{case result of } \{\text{inl } y. \text{raise}_x x'; \text{inr } x'. \text{return } y\}) \end{aligned}$$

where  $\text{translate}(M)$  is implicitly weakened by adding  $x$  to  $\Delta$ .

**Theorem 3.7 (translation preserves semantics)**

- (i) Let  $\Gamma \models M : A$  a term of the language with sum-based iteration, and  $\rho \in \llbracket \Gamma \rrbracket$ .  
Then  $\llbracket M \rrbracket_{\rho} = \llbracket \text{translate}(M) \rrbracket_{\rho}$ .
- (ii) Let  $\Gamma \models V : A$  a value of the language with sum-based iteration, and  $\rho \in \llbracket \Gamma \rrbracket$ .  
Then  $\llbracket V \rrbracket_{\rho} = \llbracket \text{translate}(V) \rrbracket_{\rho}$ .

**Corollary 3.8** If  $M \Downarrow T$  in the language with sum-based iteration, then there is  $T'$  such that  $\text{translate}(M) \Downarrow T'$  in the language with labelled iteration, and  $\llbracket T \rrbracket_{\emptyset} = \llbracket T' \rrbracket_{\emptyset}$ . And if  $M$  does not reduce to a terminal, then  $\text{translate}(M)$  does not reduce to a terminal.

## 4 Discussion and related work

In our presentation of labelled iteration, we have chosen to only consider pure functions. It is an important future task to extend the present system so as to allow for functions that raise an iteration.

Many programming languages have not just unlabelled and labelled `continue`, after which we have modelled our combination of `iter` and `raise`, but also unlabelled and labelled `break`. It should be straightforward to introduce a construct that binds a label like `iter`, but when the label is raised with parameter  $a$ , the result of that construct is  $a$ , so that `raise`

of that label resembles `break`. Such a construct resembles an intra-procedural form of exceptions. If we wrap an `iter` inside this new construct, we can “`break`” and “`continue`” from this combination of constructs, to deepen the resemblance with Java-style loops.

We have noticed the *De Bruijn index problem* in settings other than iteration. For instance, it is customary in functional languages such as Haskell to use monad transformers [12] to embed imperative programs with multiple side-effects, but they suffer from a similar De Bruijn index problem: the  $i^{\text{th}}$  monad transformer is addressed by writing “`lifti effect`”. This problem and proposed solutions have been studied in the literature [15, 24, 5], but addressing effects using labels seems yet unexplored. Imperative languages address mutable cells using identifiers, and it is possible that addressing effects with labels might benefit the readability of similar functional programs as well.

## 5 Conclusion

In the present article we summarize the essence of the sum-based representation of iteration, and evaluate it from a programming perspective. Although it might work well for a semantics standpoint, it is inadequate for programmers to program in. We propose an alternative representation of iteration that is suitable for programmers, but still has relatively clean semantics.

## References

- [1] Peter Aczel, Jiří Adámek, Stefan Milius, and Jiří Velebil. Infinite trees and completely iterative theories: a coalgebraic view. *Theoretical Computer Science*, 300(1–3):1–45, May 2003.
- [2] Jiří Adámek, Stefan Milius, and Jiří Velebil. Elgot Algebras. *Logical Methods in Computer Science*, 2(5), November 2006.
- [3] Jiří Adámek, Stefan Milius, and Jiří Velebil. Elgot theories: a new perspective on the equational properties of iteration. *Mathematical Structures in Computer Science*, 21(Special Issue 02):417–480, April 2011.
- [4] Stefan Berghofer and Christian Urban. A Head-to-Head Comparison of de Bruijn Indices and Names. *Electronic Notes in Theoretical Computer Science*, 174(5):53–67, June 2007.
- [5] Edwin Brady. Programming and Reasoning with Algebraic Effects and Dependent Types. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP ’13, pages 133–144, New York, NY, USA, 2013. ACM.
- [6] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, January 1972.
- [7] Calvin C. Elgot. Monadic Computation and Iterative Algebraic Theories. In H. E. Rose and J. C. Shepherdson, editors, *Studies in Logic and the Foundations of Mathematics*, volume 80 of *Logic Colloquium ’73 Proceedings of the Logic Colloquium*, pages 175–230. Elsevier, 1975.
- [8] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *14th Symposium on Logic in Computer Science, 1999. Proceedings*, pages 193–202, 1999.
- [9] Sergey Goncharov, Christoph Rauch, and Lutz Schröder. Unguarded Recursion on Coinductive Resumptions. In *Proceedings of the 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI)*, volume 319 of *Electronic Notes in Theoretical Computer Science*, pages 183–198. Elsevier, December 2015.
- [10] Sergey Goncharov, Lutz Schröder, and Till Mossakowski. Kleene Monads: Handling Iteration in a Framework of Generic Effects. In Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki, editors, *Algebra and Coalgebra in Computer Science*, number 5728 in *Lecture Notes in Computer Science*, pages 18–33. Springer Berlin Heidelberg, September 2009. DOI: 10.1007/978-3-642-03741-2\_3.
- [11] James Gosling, Bill Joy, Guy L. Steele, Gilad Bracha, and Alex Buckley. *The Java® language specification*. Addison-Wesley, Upper Saddle River, NJ, java se 8 edition, 2014.

- [12] Mark P. Jones. Functional Programming with Overloading and Higher-Order Polymorphism. In Johan Jeuring and Erik Meijer, editors, *Advanced Functional Programming*, volume 925, pages 97–136. Springer, 1995.
- [13] Yoshihiko Kakutani. Duality between Call-by-Name Recursion and Call-by-Value Iteration. In Julian Bradfield, editor, *Computer Science Logic*, number 2471 in Lecture Notes in Computer Science, pages 506–521. Springer Berlin Heidelberg, September 2002. DOI: 10.1007/3-540-45793-3\_34.
- [14] Fairouz Kamareddine and Alejandro Ríos. A  $\lambda$ -calculus à la de Bruijn with explicit substitutions. In Manuel Hermenegildo and S. Doaitse Swierstra, editors, *Programming Languages: Implementations, Logics and Programs*, number 982 in Lecture Notes in Computer Science, pages 45–62. Springer Berlin Heidelberg, September 1995. DOI: 10.1007/BFb0026813.
- [15] Ohad Kammar, Sam Lindley, and Nicolas Oury. Handlers in Action. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming, ICFP '13*, pages 145–158, New York, NY, USA, 2013. ACM.
- [16] Andrew Kennedy. Compiling with Continuations, Continued. In *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming, ICFP '07*, pages 177–190, New York, NY, USA, 2007. ACM.
- [17] Dexter Kozen and Konstantinos Mamouras. Kleene Algebra with Products and Iteration Theories. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 415–431, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
- [18] Dexter Kozen and Konstantinos Mamouras. Kleene Algebra with Equations. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, number 8573 in Lecture Notes in Computer Science, pages 280–292. Springer Berlin Heidelberg, July 2014. DOI: 10.1007/978-3-662-43951-7\_24.
- [19] J. Laird. The Elimination of Nesting in SPCF. In Paweł Urzyczyn, editor, *Typed Lambda Calculi and Applications*, number 3461 in Lecture Notes in Computer Science, pages 234–245. Springer Berlin Heidelberg, April 2005. DOI: 10.1007/11417170\_18.
- [20] Paul Blain Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, 19(4):377–414, December 2006.
- [21] John Longley. The recursion hierarchy for PCF is strict. Informatics Research Report EDI-INF-RR-1421, School of Informatics, University of Edinburgh, July 2015.
- [22] Conor McBride and James McKinna. Functional Pearl: I Am Not a Number—I Am a Free Variable. In *Proceedings of the 2004 ACM SIGPLAN Workshop on Haskell, Haskell '04*, pages 1–9, New York, NY, USA, 2004. ACM.
- [23] Stefan Milius and Tadeusz Litak. Guard Your Daggers and Traces: On The Equational Properties of Guarded (Co-)recursion. *Electronic Proceedings in Theoretical Computer Science*, 126:72–86, August 2013.
- [24] Dominic Orchard and Tomas Petricek. Embedding Effect Systems in Haskell. In *Proceedings of the 2014 ACM SIGPLAN Symposium on Haskell, Haskell '14*, pages 13–24, New York, NY, USA, 2014. ACM.
- [25] W. W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(02):198–212, August 1967.

## A Appendix: proofs

We first prove adequacy of fine-grain call-by-value without iteration. The adequacy of FGCBV + sum-based iteration and the adequacy of the language with labelled iteration are then minor modifications. All our adequacy proofs are in the style of Tait [25].

We use the following substitution lemma for both plain FGCBV and FGCBV with sum-based iteration.

**Lemma A.1** *Assume a substitution  $\sigma : \Gamma' \rightarrow \Gamma$  and an environment  $\rho \in \llbracket \Gamma \rrbracket$ .*

- (i) *Let  $\Gamma' \models V : A$  be a value. Then  $\llbracket V \rrbracket_{(x \mapsto \llbracket \sigma x \rrbracket_\rho)_{x \in \Gamma'}} = \llbracket V \sigma \rrbracket_\rho$ .*
- (ii) *Let  $\Gamma' \models M : A$  be a term. Then  $\llbracket M \rrbracket_{(x \mapsto \llbracket \sigma x \rrbracket_\rho)_{x \in \Gamma'}} = \llbracket M \sigma \rrbracket_\rho$ .*

The proofs of both substitution lemmas, Lemma A.1 and Lemma 3.5, are routine and we omit them.

### A.1 Adequacy of FGCBV without iteration

We prove adequacy with the help of the following type-indexed predicate on closed values and terms.

**Definition A.2** By mutual induction on the type of  $V$  and  $M$ , respectively.

when $\vdash^V V : 1 :$	$P(V) \equiv \text{true}$
when $\vdash^V V : \text{nat} :$	$P(V) \equiv \text{true}$
when $\vdash^V V : A + B :$	$P(\text{inl } V) \equiv P(V)$ $P(\text{inr } V) \equiv P(V)$
when $\vdash^V V : A \times B :$	$P(\langle V, W \rangle) = P(V) \wedge P(W)$
when $\vdash^V V : A \rightarrow B :$	$P(\lambda x. M) \equiv \forall (\vdash^V W : A) : P(W) \Rightarrow P(M[W/x])$
when $\vdash^E M : A :$	$P(M) \equiv \exists (\vdash^V V : A) : (P(V) \wedge M \Downarrow \text{return } V \wedge \llbracket M \rrbracket_{\emptyset} = \text{inl } \llbracket V \rrbracket_{\emptyset})$

Observe that  $P(M)$  implies adequacy of  $M$ .

#### Proposition A.3

- (i) If  $\Gamma \vdash^V V : A$ , and if for all  $(x:B) \in \Gamma$  we have a closed  $\vdash^V \sigma_x : B$  satisfying  $P(\sigma_x)$ , then  $P(V\sigma)$ .
- (ii) If  $\Gamma \vdash^E M : A$ , and if for all  $(x:B) \in \Gamma$  we have a closed  $\vdash^V \sigma_x : B$  satisfying  $P(\sigma_x)$ , then  $P(M\sigma)$ .

#### Proof

By induction on the value or term. Here are some interesting and less interesting cases.

$V = x$ ) Then  $V\sigma = \sigma_x$ , which was assumed to satisfy  $P$ .

$M = \text{return } V$ ) Trivially by induction.

$V = \lambda y. M$ ) We have to show that if  $\vdash^V W : A$  satisfies  $P$ , then  $M[\sigma, W/y]$  satisfies  $P$ . By induction.

$M = \text{let } V \text{ be } x. N$ ) We are allowed to assume  $P(V\sigma)$ , so the induction hypothesis gives us  $P(N[\sigma, (V\sigma)/x])$ . We know that  $M\sigma$  and  $N[\sigma, (V\sigma)/x]$  reduce to the same terminal. We know  $\llbracket M\sigma \rrbracket_{\emptyset} = \llbracket N\sigma \rrbracket_{x \mapsto \llbracket V\sigma \rrbracket_{\emptyset}}$ , which we know is equal to  $\llbracket N[\sigma, (V\sigma)/x] \rrbracket_{\emptyset}$  by the substitution lemma. Now  $P(N[\sigma, (V\sigma)/x])$  implies  $P(M\sigma)$ .

$M = V W$ ) Similarly.

$M = M' \text{ to } x. N$ ) From the induction, we get  $V$  such that  $P(V)$  and  $M'\sigma \Downarrow \text{return } V$  and  $\llbracket M'\sigma \rrbracket_{\emptyset} = \text{inl } \llbracket V \rrbracket_{\emptyset}$ . From the derivation rule and the induction, we get  $V'$  such that  $P(V')$  and  $N[\sigma, V/x] \Downarrow \text{return } V'$ , and  $\llbracket N[\sigma, V/x] \rrbracket_{\emptyset} = \text{inl } \llbracket V' \rrbracket_{\emptyset}$ .

By the substitution lemma,  $\llbracket N\sigma[V/x] \rrbracket_{\emptyset} = \llbracket N\sigma \rrbracket_{x \mapsto \llbracket V \rrbracket_{\emptyset}}$ , and because we know  $\llbracket M'\sigma \rrbracket_{\emptyset} = \text{inl } \llbracket V \rrbracket_{\emptyset}$ , we know that by definition

$$\llbracket (M'\sigma) \text{ to } x. (N\sigma) \rrbracket_{\emptyset} = \llbracket N\sigma \rrbracket_{x \mapsto \llbracket V \rrbracket_{\emptyset}}.$$



This completes the proof for this case.

$M = \text{case } V \text{ of } \dots$ ) Depending on the type of  $V$ , but for every type trivially by case analysis on  $V\sigma$ .

□

**Corollary A.4** *All closed values and terms satisfy  $P$ .*

Adequacy directly follows from this.

Observe that the only cases in which we essentially looked at the normal form of  $M\sigma$  are return  $V$  and  $M$  to  $x$ .  $N$ . Specifically, we did not use the normal form of  $M\sigma$  in the let case. This means that we can reuse most of the proof for FGCBV with sum-based iteration.

### A.2 Adequacy of FGCBV + sum-based iteration

Similar structure. We redefine  $P(M)$ :

$$P(\vdash M : A) = \exists(\vdash V : A) : \left( (P(V) \wedge M \Downarrow \text{return } V \wedge \llbracket M \rrbracket_{\emptyset} = \text{inl } \llbracket V \rrbracket_{\emptyset}) \right. \\ \left. \vee (M \Downarrow \wedge \llbracket M \rrbracket_{\emptyset} = \text{inr } \perp) \right)$$

We have the same proposition as Proposition A.3 in this case:

- The case  $M = \text{return } V$  is still trivial.
- For  $M = M' \text{ to } x. N$ , we have to consider the alternative case that  $M'\sigma \Downarrow$  and  $\llbracket M'\sigma \rrbracket_{\emptyset} = \text{inr } \perp$ . This case is trivial.
- For iter, observe that every sequence  $V_1, \dots, V_k$  in the operational semantics corresponds uniquely to a sequence

$$v_0 = \llbracket V\sigma \rrbracket_{\emptyset}, v_1 = \llbracket V_1 \rrbracket_{\emptyset}, v_2 = \llbracket V_2 \rrbracket_{\emptyset}, \dots, v_k = \llbracket V_k \rrbracket_{\emptyset}$$

for the denotational semantics, and the proof in that case is analogous to the proof for let.

To prove that non-existence of a valid sequence  $V_1, \dots, V_k$  for the operational semantics implies the non-existence of a valid sequence  $v_0, \dots, v_k$ , we instead prove the contrapositive. Indeed, we have our initial  $V\sigma$  already, and by induction on a valid sequence  $v_0, \dots, v_k$  together with the induction hypothesis, we obtain step by step our sequence  $V_1, \dots, V_k$ . So now we also know that  $\text{iter } V, x.M \Downarrow$  implies  $\llbracket \text{iter } V, x.M \rrbracket_{\emptyset} = \text{inr } \perp$ .

### A.3 Adequacy of the language with labelled iteration

Similar structure.

We redefine  $P(M)$  again. Recall that  $M$  closed means that  $\Delta; \cdot \vdash^c M : A$ .

$$P(M) \equiv \left( \begin{aligned} & \left( \exists (\vdash^v V : A) : \left( P(V) \wedge M \Downarrow \text{return } V \wedge \llbracket M \rrbracket_\emptyset = \text{return } \llbracket V \rrbracket_\emptyset \right) \right) \\ & \vee \left( \exists ((x:B) \in \Delta) : \exists (\vdash^v V : B) : \left( P(V) \wedge M \Downarrow \text{raise}_x V \wedge \llbracket M \rrbracket_\emptyset = \text{raise}_x \llbracket V \rrbracket_\emptyset \right) \right) \\ & \vee \left( M \Downarrow \wedge \llbracket M \rrbracket_\emptyset = \perp \right) \end{aligned} \right)$$

We have a proposition analogous to Proposition A.3.

**Proposition A.5**

- (i) *If  $\Gamma \vdash^v V : A$ , and if for all  $(x:B) \in \Gamma$  we have a closed  $\vdash^v \sigma_x : B$  satisfying  $P(\sigma_x)$ , then  $P(V\sigma)$ .*
- (ii) *If  $\Delta; \Gamma \vdash^c M : A$ , and if we have a substitution  $\sigma : \Delta'; \Gamma' \rightarrow \Delta; \cdot$  such that  $P(\sigma_{\text{id}}(x))$  on all identifiers, then  $P(M\sigma)$ .*

- The additional case for sequencing is trivial.
- The case  $P(\text{raise}_x V)$  is trivial.
- The additional case for iteration is analogous.

# Giry and the Machine

Fredrik Dahlqvist<sup>1</sup> Vincent Danos<sup>2</sup> Ilias Garnier<sup>3</sup>

*University College London*

*Ecole Normale Supérieure*

*University of Edinburgh*

---

## Abstract

We present a general method - the Machine - to analyse and characterise in finitary terms natural transformations between (iterates of) Giry-like functors in the category **Pol** of Polish spaces. The method relies on a detailed analysis of the structure of **Pol** and a small set of categorical conditions on the domain and codomain functors. We apply the Machine to transformations from the Giry and positive measures functors to combinations of the Vietoris, multiset, Giry and positive measures functors. The multiset functor is shown to be defined in **Pol** and its properties established. We also show that for some combinations of these functors, there cannot exist more than one natural transformation between the functors, in particular the Giry monad has no natural transformations to itself apart from the identity. Finally we show how the Dirichlet and Poisson processes can be constructed with the Machine.

*Keywords:* probability, topology, category theory, monads

---

## 1 Introduction

Classical tools of probability theory are not geared towards compositionality, and especially not compositional approximation (Kozen, [13]). This has not prevented authors from developing powerful techniques (Chaput et al. [5], Kozen et al. [14]) based on structural approaches to probability theory (Giry, [9]). Here, we adopt a slightly different standpoint: we propose to tackle this tooling problem *globally*, by combining structural insights of **Pol** together with some classical tools of probability theory and topology put in functorial form. The outcome is the Machine, an axiomatic reconstruction in category-theoretic terms of developments carried out in [7]. Thus, we get a simpler and more conceptual proof of our previous results. We also obtain a much more comprehensive picture and prove that natural transformations between Giry-like functors are entirely characterised by their components on finite spaces. For instance, the monadic data of the Giry functor are easily obtained

---

This work was sponsored by the European Research Council (ERC) under the grant RULE (320823).

<sup>1</sup> [f.dahlqvist@ucl.ac.uk](mailto:f.dahlqvist@ucl.ac.uk)

<sup>2</sup> [vincent.danos@ens.fr](mailto:vincent.danos@ens.fr)

<sup>3</sup> [igarnier@inf.ed.ac.uk](mailto:igarnier@inf.ed.ac.uk)

from the finite case (which is completely elementary) and applying the Machine. But the construction is not limited to probability functors: we deal similarly with the multiset and the Vietoris (the topological powerdomain of compact subsets) functors. This allows one to consider transformations mixing probabilistic and ordinary non-determinism, in a way which is reminiscent of (Keimel et al., [12]). Another byproduct of our Machine is that we reconstruct from finitary data classical objects of probability theory and statistics, namely the Poisson and Dirichlet processes. It is worth noting that Poisson, Dirichlet (and many other similar constructions obtained by recombining the basic ingredients differently) are obtained as natural and continuous maps: naturality expresses the stability of the “behaviour” in a change of granularity, and as such is a fundamental property of consistency, but continuity (which to our knowledge is proved here for the first time) expresses a no less important property, namely the robustness of the behaviour in changes in “parameters”. This has potential implications in Bayesian learning.

The structure of the paper is as follows. In Sec. 3, we show that **Pol** is stratified into the subcategories **Pol<sub>f</sub>**, **Pol<sub>cz</sub>**, **Pol<sub>z</sub>** of finite, compact zero-dimensional and zero-dimensional Polish spaces respectively and show how these subcategories are related. In Sec. 4, the Machine is introduced: we identify a small set of categorical conditions on functors  $F, G$  that guarantee that any natural transformation from  $F$  to  $G$  in **Pol<sub>f</sub>** can be extended step-by-step through the subcategories to a natural transformation on **Pol**. In Sec. 5, we illustrate the workings of the Machine on natural transformations connecting the Giry and positive measure functors to combinations of the Vietoris, multiset, Giry and positive measure functors. As far as we know, the multiset functor is defined in **Pol** for the first time and its properties are established. As a first application of the Machine, we develop in Sec. 6 general criteria under which there can exist at most one natural transformation from a functor  $F$  to the Giry functor. In particular, we show that there exists at most one natural transformation between the Vietoris, multiset, positive measure and Giry functor to the Giry functor. Lastly, we show in Sec. 7 how transformations of the type  $M^+ \Rightarrow GH$  where  $M^+$  is the finite measure functor and  $H$  is either the multiset or the finite measure functor can be built in **Pol<sub>f</sub>** from a single generating morphism  $M^+(1) \rightarrow GH(1)$  and give criteria for this transformation to be natural. In particular, we show that the Dirichlet and Poisson distributions satisfy these criteria and use the Machine to build Dirichlet and Poisson processes.

To save space and ease reading, several proofs are given after the main text in appendices.

## 2 Notations

Most of our developments take place in the category **Pol** of Polish spaces and continuous maps. **Pol** is a full subcategory of the category **Top** of topological spaces and continuous maps. **Pol** has all countable limits and all countable coproducts (Bourbaki [4], IX). The functor mapping any space to the measurable space having the same underlying set and the Borel  $\sigma$ -algebra and interpreting continuous maps as measurable ones will be denoted by  $\mathcal{B} : \mathbf{Pol} \rightarrow \mathbf{Meas}$ , where **Meas** is the category of measurable spaces and measurable maps. A *countable codirected diagram* (*ccd* for

short) is given by a countable directed partial order (DPO)  $\mathcal{I}$  and a contravariant functor  $D : \mathcal{I}^{op} \rightarrow \mathbf{Pol}$  such that for all  $i \leq_{\mathcal{I}^{op}} j$ ,  $D(i \leq_{\mathcal{I}^{op}} j)$  is surjective. We moreover assume that *ccds* range over non-empty spaces. With that assumption, the categorical limit of a *ccd*  $D$ , which we denote by  $\lim D$ , is always non-empty.

### 3 The structure of $\mathbf{Pol}$

$\mathbf{Pol}$  can be decomposed according to the following diagram of inclusions:

$$\mathbf{Pol}_f \xrightarrow{I_{cz}} \mathbf{Pol}_{cz} \xrightarrow{I_z} \mathbf{Pol}_z \xrightarrow{I_p} \mathbf{Pol} \quad (1)$$

Here,  $\mathbf{Pol}_f$  is the full subcategory of finite (hence discrete) spaces,  $\mathbf{Pol}_{cz}$  is the full subcategory of compact zero-dimensional spaces and  $\mathbf{Pol}_z$  is the full subcategory of zero-dimensional spaces while  $I_{cz}, I_z$  and  $I_p$  are the obvious inclusion functors. To this picture, we add categories of *based spaces* and *base-preserving maps*.

**Definition 3.1 (Categories of based spaces)** A *based space* is a pair  $(X, \mathcal{F})$  of  $X \in \text{Obj}(\mathbf{Pol})$  and of a countable base  $\mathcal{F}$  of the topology of  $X$ . A *base-preserving map* from  $(X, \mathcal{F})$  to  $(Y, \mathcal{G})$  is a function  $f : X \rightarrow Y$  such that  $f^{-1}(\mathcal{G}) \subseteq \mathcal{F}$  (it is therefore continuous). One easily checks that this defines a category having based spaces as objects and base-preserving maps as morphisms. We denote this category by  $\mathbf{Pol}^b$ . Similarly, a *based zero-dimensional space* is a pair  $(Z, \mathcal{F})$  where  $Z \in \text{Obj}(\mathbf{Pol}_z)$  and  $\mathcal{F}$  is a *countable base of clopen sets which is also a boolean algebra*. We denote by  $\mathbf{Pol}_z^b$  the category of based zero-dimensional spaces and base-preserving maps.

Of course, there exists for each such based category a (faithful, but not full!) forgetful functor, that we will denote by resp.  $U_z$  and  $U_p$ . The situation is summed up in the following commutative diagram in **Cat**:

$$\begin{array}{ccccc} & & & \mathbf{Pol}_z^b & \xrightarrow{I_p^b} & \mathbf{Pol}^b \\ & \nearrow I_z^b & & \downarrow U_z & & \downarrow U_p \\ \mathbf{Pol}_f & \xrightarrow{I_{cz}} & \mathbf{Pol}_{cz} & & & \\ & \searrow I_z & & \mathbf{Pol}_z & \xrightarrow{I_p} & \mathbf{Pol} \end{array}$$

In the remainder of this section, we will unravel further relationships between these categories.

**$\mathbf{Pol}_f$  is a codense subcategory of  $\mathbf{Pol}_{cz}$ .** Objects of  $\mathbf{Pol}_f$  are finite discrete spaces. Note that every subset of a discrete space is clopen; as a consequence, any map between two finite spaces is continuous. We will denote objects of  $\mathbf{Pol}_f$  by their cardinality  $m, n$ . The objects of  $\mathbf{Pol}_{cz}$  are the compact zero-dimensional (or *profinite*) spaces, a prime example being the Cantor space  $2^{\mathbb{N}}$ . These spaces are homeomorphic to limits of countable codirected diagrams (*ccds* for short) taking values in  $\mathbf{Pol}_f$ . This is exactly captured by the concept of codensity (see [15], X.6).

**Proposition 3.2**  *$\mathbf{Pol}_f$  is codense in  $\mathbf{Pol}_{cz}$ .*

**Proof.** Let  $X$  be a compact zero-dimensional space, and consider the comma category  $X \downarrow I_{cz}$ . We denote by  $D_X : (X \downarrow I_{cz}) \rightarrow \mathbf{Pol}_f$  the diagram corresponding to the base of this cone. It is enough to prove that for all  $X \in \text{Obj}(\mathbf{Pol}_{cz})$ ,  $X \cong \lim D_X$ . Following (Mac Lane [15], IX.3), it is in turn enough to exhibit a diagram  $D : \mathcal{I}^{op} \rightarrow \mathbf{Pol}_f$  verifying  $X \cong \lim D$  and a cofinal (“initial” in [15]) functor  $c : \mathcal{I}^{op} \rightarrow (X \downarrow I_{cz})$ . Proposition 3.1 of [7] yields the existence of such a diagram  $D$  where  $\mathcal{I}$  is the set of finite partitions of  $X$  taken in the boolean algebra of clopen sets of  $X$  (that we denote by  $Clo(X)$ ), partially ordered by partition refinement and directed by partition intersection. Observe that any continuous map  $f : X \rightarrow n$  induces a finite clopen partition of  $X$  by considering its fibres. Let us denote this partition by  $X/f$ . Let  $c$  be the functor mapping any finite partition  $n \in \mathcal{I}^{op}$  seen as an object of  $\mathbf{Pol}_f$  to the quotient map  $q_n : X \twoheadrightarrow n$ , and any refinement  $m \leq_{\mathcal{I}^{op}} n$  to the obvious map  $\pi_{mn}$  such that  $q_m = \pi_{mn} \circ q_n$ . For any  $f : X \rightarrow n$  the partition  $X/f$  is mapped to  $c(X/f) : X \rightarrow X/f$ , and there trivially exists a map  $\pi : c(X/f) \rightarrow f$ . For any two  $f, f' \in \text{Obj}(X \downarrow I_{cz})$ , one can easily exhibit a partition  $i \in \mathcal{I}$  of  $X$  such that there exists  $\pi : c(i) \rightarrow f$  and  $\pi' : c(i) \rightarrow f'$ .  $\square$

**$\mathbf{Pol}_{cz}$  is a reflective subcategory of  $\mathbf{Pol}_z^b$ .** Objects of  $\mathbf{Pol}_z$  are *zero-dimensional spaces*, i.e. spaces whose topology admits a (countable) base of clopen sets. Discrete spaces (such as  $\mathbb{N}$ ) are always zero-dimensional. A less trivial example is the Baire space  $\mathbb{N}^{\mathbb{N}}$ . The bridge between  $\mathbf{Pol}_{cz}$  and  $\mathbf{Pol}_z$  is provided by *compactifying* zero-dimensional spaces, as explained in full length in ([7], Sec. 3). Let us recall the underpinnings of this compactification. Let  $Z$  be some zero-dimensional space and  $\mathcal{F}$  be a countable base of clopens of  $Z$ . One easily verifies that the boolean algebra generated by  $\mathcal{F}$ , that we denote by  $Bool(\mathcal{F})$ , still generates the same topology and is still countable. Therefore, one can without loss of generality assume that the base  $\mathcal{F}$  of  $Z$  is a countable Boolean algebra of clopen sets (that we call a *boolean base* for short). Let  $\mathcal{I}_{\mathcal{F}}$  be the directed partial order of finite partitions of  $Z$  taken in  $\mathcal{F}$  and let  $D_{\mathcal{F}} : \mathcal{I}_{\mathcal{F}}^{op} \rightarrow \mathbf{Pol}_f$  be the diagram defined by  $D_{\mathcal{F}}(i \in \mathcal{I}_{\mathcal{F}}^{op}) \triangleq i$  on objects (seeing finite partitions of  $Z$  as finite discrete spaces) and  $D_{\mathcal{F}}(j \leq_{\mathcal{I}_{\mathcal{F}}^{op}} i) = q_{ij}$  where  $q_{ij} : j \rightarrow i$  is the obvious quotient map.

**Proposition 3.3 (Wallman compactification ([7], Prop. 3.12))**  *$\lim D_{\mathcal{F}}$  is a zero-dimensional compactification of  $Z$  that we denote by  $\omega_{\mathcal{F}}(Z)$ . We denote by  $\eta_{\mathcal{F}} : Z \hookrightarrow \omega_{\mathcal{F}}(Z)$  the canonical embedding of  $Z$  into its compactification.*

Note that this compactification is not universal, in the sense that  $\mathbf{Pol}_{cz}$  is not a reflective subcategory of  $\mathbf{Pol}_z$  (see [15], IV.3 for a definition of reflective subcategory). However, we will show that  $\mathbf{Pol}_{cz}$  is a reflective subcategory of  $\mathbf{Pol}_z^b$ . In the following, recall that  $Clo(X)$  is the boolean algebra of clopen sets of a compact zero-dimensional space  $X$ .

**Proposition 3.4** *Let  $I_z^b$  be the operation that maps any compact zero-dimensional space  $X$  to the pair  $(X, Clo(X))$  and which acts identically on maps between such spaces.  $I_z^b$  is a full and faithful functor from  $\mathbf{Pol}_{cz}$  to  $\mathbf{Pol}_z^b$ .*

**Proof.** For any space  $X \in \text{Obj}(\mathbf{Pol}_{cz})$ , its boolean algebra of clopen sets  $Clo(X)$  is countable and therefore,  $(X, Clo(X))$  is a based zero-dimensional space. By continuity, maps between such spaces are base-preserving. Functoriality, fullness and

faithfulness are trivial.  $\square$

Our compactification naturally lives in  $\mathbf{Pol}_z^b$ :

**Proposition 3.5** *For any  $(Z, \mathcal{F}) \in \text{Obj}(\mathbf{Pol}_z^b)$ , the embedding  $\eta_{\mathcal{F}} : (Z, \mathcal{F}) \rightarrow I_z^b(\omega_{\mathcal{F}}(Z))$  is base preserving.*

**Proof.** By construction of  $\omega_{\mathcal{F}}(Z)$ , any finite clopen partition of this space will induce through  $\eta_{\mathcal{F}}$  a finite partition of  $Z$  taken in  $\mathcal{F}$ . Therefore,  $\eta_{\mathcal{F}}$  is base preserving.  $\square$

The following proposition states the functoriality of compactification in this new setting, and the fact that  $\mathbf{Pol}_{cz}$  is a reflective subcategory of  $\mathbf{Pol}_z^b$ .

**Proposition 3.6 ( $\omega$  as a reflector)** (i) *Let  $f : (Z, \mathcal{F}) \rightarrow (Z', \mathcal{F}')$  be a base-preserving map. There exists a unique  $\omega_{\mathcal{F}\mathcal{F}'}(f) : \omega_{\mathcal{F}}(Z) \rightarrow \omega_{\mathcal{F}'}(Z')$  such that  $\omega_{\mathcal{F}\mathcal{F}'}(f) \circ \eta_{\mathcal{F}} = \eta_{\mathcal{F}'} \circ f$ . (ii)  $\omega : \mathbf{Pol}_z^b \rightarrow \mathbf{Pol}_{cz}$  is a functor defined on objects by  $\omega(Z, \mathcal{F}) \triangleq \omega_{\mathcal{F}}(Z)$  and on base-preserving maps  $f : (Z, \mathcal{F}) \rightarrow (Z', \mathcal{F}')$  by  $\omega(f) \triangleq \omega_{\mathcal{F}\mathcal{F}'}(f)$ , and it is left adjoint to the inclusion functor  $I_z^b$  (the unit being given by  $\eta$ ).*

**Proof.** (i) This is Prop. 3.13 and Corollary 3.14 of [7]. Let us sketch the argument. As  $f$  is base-preserving, any finite clopen partition of  $Z'$  taken in  $\mathcal{F}'$  will induce a unique finite clopen partition of  $Z$  taken in  $\mathcal{F}$ . Using the notations of Prop. 3.3, we deduce that  $D_{\mathcal{F}'}$  is a sub-diagram of  $D_{\mathcal{F}}$ . Therefore, there exists a unique mediating map (that we denote  $\omega_{\mathcal{F}\mathcal{F}'}(f)$ ) from  $\lim D_{\mathcal{F}}$  to  $\lim D_{\mathcal{F}'}$ , i.e. from  $\omega_{\mathcal{F}}(Z)$  to  $\omega_{\mathcal{F}'}(Z')$ , such that  $\eta_{\mathcal{F}'} \circ f = \omega_{\mathcal{F}\mathcal{F}'}(f) \circ \eta_{\mathcal{F}}$ . (ii)  $\omega$  trivially preserves identities. For all  $f, f'$ , the equality  $W(f' \circ f) = W(f') \circ W(f)$  is a consequence of the uniqueness of factorisations in (i). According to (Mac Lane [15], IV.3), left adjointness of  $\omega$  is a direct consequence of (i), as any map  $f : (Z, \mathcal{F}) \rightarrow I_z^b(X)$  will factor uniquely through  $\eta_{\mathcal{F}} : (Z, \mathcal{F}) \rightarrow I_z^b(\omega_{\mathcal{F}}(Z))$ .  $\square$

This reflection is summarised in the following diagram:

$$\begin{array}{ccccc}
 & & Id_{\mathbf{Pol}_z^b} & & \\
 & \swarrow & \Downarrow \eta & \searrow & \\
 \mathbf{Pol}_z^b & \xrightarrow{\omega} & \mathbf{Pol}_{cz} & \xrightarrow{I_z^b} & \mathbf{Pol}_z^b
 \end{array} \tag{2}$$

$\mathbf{Pol}_z^b$  is a coreflective subcategory of  $\mathbf{Pol}^b$ . The penultimate step in our structural analysis of  $\mathbf{Pol}$  is to relate  $\mathbf{Pol}_z^b$  and  $\mathbf{Pol}^b$ . This is accomplished by associating zero-dimensional refinements to arbitrary spaces, in an operation called *zero-dimensionalisation*. Let us define this operation.

**Proposition 3.7 (Zero-dimensionalisation ([7], Prop. 3.2))** *Let  $X$  be a space with underlying set  $U(X)$  and let  $\mathcal{F}$  be a countable base of  $X$ . The topological space  $z_{\mathcal{F}}(X) \triangleq (U(X), \langle \text{Bool}(\mathcal{F}) \rangle)$  having as underlying set  $U(X)$  and whose topology is generated by the boolean algebra  $\text{Bool}(\mathcal{F})$  verifies the following properties:*

- (i)  $z_{\mathcal{F}}(X)$  is Polish;

- (ii)  $z_{\mathcal{F}}(X)$  is zero-dimensional.
- (iii) measurable sets are preserved:  $\mathcal{B}(X) = \mathcal{B}(z_{\mathcal{F}}(X))$ .

In a similar fashion to compactifications, this operation is better typed as a functor from  $\mathbf{Pol}^b$  to  $\mathbf{Pol}_z^b$ . Let us make zero-dimensionalisation into a functor:

**Proposition 3.8** *Let  $f : (X, \mathcal{F}) \rightarrow (Y, \mathcal{G})$  be a base-preserving map in  $\mathbf{Pol}^b$ . Then  $f : (z_{\mathcal{F}}(X), \text{Bool}(\mathcal{F})) \rightarrow (z_{\mathcal{G}}(Y), \text{Bool}(\mathcal{G}))$  is base-preserving in  $\mathbf{Pol}_z^b$ . We denote by  $z : \mathbf{Pol}^b \rightarrow \mathbf{Pol}_z^b$  the functor defined by  $z(X, \mathcal{F}) = (z_{\mathcal{F}}(X), \text{Bool}(\mathcal{F}))$  on objects and acting identically on arrows.*

**Proof.** It is sufficient to consider the case of an arbitrary finite union of literals  $L = A_1^{\epsilon_1} \cup \dots \cup A_n^{\epsilon_n} \in \text{Bool}(\mathcal{G})$ , where  $A_i \in \mathcal{G}$  and  $A_i^{\epsilon_i}$  denotes either  $A_i^c$  or  $A_i$ . We have  $f^{-1}(L) = \cup_{i=1}^n f^{-1}(A_i)^{\epsilon_i}$ , since  $f$  is base-preserving in  $\mathbf{Pol}^b$  we deduce that  $f^{-1}(L) \in \text{Bool}(\mathcal{F})$ . Continuity of  $f$  in  $\mathbf{Pol}_z^b$  is a direct consequence of base preservation. The fact that  $z$  is a functor is now trivial.  $\square$

The following result now follows easily:

**Proposition 3.9 ( $z$  as a coreflector)**  *$z$  is right adjoint to the inclusion functor  $I_p^b$ , i.e.  $\mathbf{Pol}_z^b$  is a coreflective subcategory of  $\mathbf{Pol}^b$ .*

**Proof.** Observe that for all  $(X, \mathcal{F}) \in \text{Obj}(\mathbf{Pol}^b)$ , the identity function  $\epsilon_{\mathcal{F}} \triangleq \text{id} : I_p^b z(X, \mathcal{F}) \rightarrow (X, \mathcal{F})$  is base-preserving. This indeed constitutes the counit of the coreflection: one easily verifies that for all  $f : I_p^b(Z, \mathcal{F}) \rightarrow (X, \mathcal{G})$  there exists a unique  $f' : I_p^b(Z, \mathcal{F}) \rightarrow I_p^b z(X, \mathcal{G})$  such that  $f = \epsilon_{\mathcal{G}} \circ f'$  (and  $f'$  is equal to  $f$  as a function).  $\square$

This coreflection is summarised in the following diagram:

$$\begin{array}{ccccc}
 & & Id_{\mathbf{Pol}^b} & & \\
 & \swarrow & \uparrow \epsilon & \searrow & \\
 \mathbf{Pol}^b & \xrightarrow{z} & \mathbf{Pol}_z^b & \xrightarrow{I_p^b} & \mathbf{Pol}^b
 \end{array} \tag{3}$$

**Relating  $\mathbf{Pol}^b$  and  $\mathbf{Pol}$ .** For all space  $X \in \text{Obj}(\mathbf{Pol})$ , let us denote the set of countable bases of  $X$ , partially ordered by inclusion, by  $Bases(X)$ . Observe that  $Bases(X)$  is directed by taking the union of the bases and closing under finite intersections. Accordingly, if  $\mathcal{F} \subseteq \mathcal{G}$  are two countable bases of  $X$ , the identity function  $\text{id} : (X, \mathcal{G}) \rightarrow (X, \mathcal{F})$  is trivially base-preserving. This defines a codirected diagram  $B_X : Bases(X)^{op} \rightarrow \mathbf{Pol}^b$  mapping any base  $\mathcal{F}$  to  $(X, \mathcal{F})$  and any pair  $\mathcal{F} \subseteq \mathcal{G}$  to the identity function. Recall that  $U_p : \mathbf{Pol}^b \rightarrow \mathbf{Pol}$  is the base-forgetting functor. The next definition and proposition provide a universal characterisation of Polish spaces in terms of their zero-dimensionalisation.

**Definition 3.10 (Diagram of zero-dimensionals)** We define the *diagram of zero-dimensionals of  $X$* :

$$Z_X \triangleq U_p I_p^b z B_X : Bases(X)^{op} \rightarrow \mathbf{Pol}$$



that maps bases  $\mathcal{F} \in \text{Bases}(X)$  to  $Z_X(\mathcal{F}) \triangleq z_{\mathcal{F}}(X)$ .

We state without proof the following result, which is a category-theoretic reformulation of ([7], Theorem 3.5):

**Proposition 3.11** *For all space  $X \in \text{Obj}(\mathbf{Pol})$ ,  $X \cong \text{colim } Z_X$ .*

In more concrete terms, any space  $X$  has the final topology for the family of identity functions  $\{id : z_{\mathcal{F}}(X) \rightarrow X\}_{\mathcal{F}}$  where  $\mathcal{F}$  ranges over  $\text{Bases}(X)$ . Let us conclude this section by summarising our structural decomposition of  $\mathbf{Pol}$  in the following diagram:

$$\begin{array}{ccccc}
 \mathbf{Pol}_f & \xrightarrow{I_{cz}} & \mathbf{Pol}_{cz} & \begin{array}{c} \xleftarrow{\omega} \\ \xrightarrow{I_z^b} \end{array} & \mathbf{Pol}_z^b & \begin{array}{c} \xleftarrow{z} \\ \xrightarrow{I_p^b} \end{array} & \mathbf{Pol}^b \\
 & & \downarrow I_z & & \downarrow U_z & & \downarrow U_p \\
 & & \mathbf{Pol}_z & \xrightarrow{I_p} & \mathbf{Pol} & & 
 \end{array} \quad (4)$$

## 4 The Machine

We will leverage the structural decomposition of  $\mathbf{Pol}$  given in the previous section to characterise some “profinite” natural transformations, in the sense that their behaviour on arbitrary spaces is entirely determined by their behaviour on finite spaces. We proceed in a stepwise and modular fashion: the Machine is presented as a series of extension theorems giving sufficient conditions for a natural transformation to be uniquely extended from a subcategory to the ambient one (Theorems 4.2-4.11). These results are combined in Theorem 4.12.

**I. From  $\mathbf{Pol}_f$  to  $\mathbf{Pol}_{cz}$ .** One can completely characterise the subcategory of the functor category  $[\mathbf{Pol}_{cz}; \mathbf{Pol}]$  consisting of functors commuting with certain codirected limits in terms of  $[\mathbf{Pol}_f; \mathbf{Pol}]$ . These functors are defined below.

**Definition 4.1 ( $\mathbf{Pol}_f$ -continuous functors)** A functor  $F : \mathbf{Pol} \rightarrow \mathbf{Pol}$  is  $\mathbf{Pol}_f$ -continuous if for all ccd  $D : \mathcal{I}^{op} \rightarrow \mathbf{Pol}_f$ ,  $F(\lim D) \cong \lim FD$ .

The key result is the following:

**Theorem 4.2** *Let  $F, G : \mathbf{Pol}_{cz} \Rightarrow \mathbf{Pol}$  be two functors. If  $G$  is  $\mathbf{Pol}_f$ -continuous, then  $\text{Nat}(F|_{\mathbf{Pol}_f}, G|_{\mathbf{Pol}_f}) \cong \text{Nat}(F, G)$ .*

This isomorphism arises from the existence of a functor computing right Kan extension along  $I_{cz}$  (see [15], X), denoted by  $\text{Ran}_{I_{cz}}$  in the following:

**Proposition 4.3** *The functor  $\text{Ran}_{I_{cz}} : [\mathbf{Pol}_f; \mathbf{Pol}] \rightarrow [\mathbf{Pol}_{cz}; \mathbf{Pol}]$  is full and faithful.*

**Proof.** In the following, for any  $X \in \text{Obj}(\mathbf{Pol}_{cz})$ ,  $D_X : (X \downarrow I_{cz}) \rightarrow \mathbf{Pol}_f$  stands for the diagram verifying  $X \cong \lim D_X$  (see proof of Prop. 3.2). We first prove that any functor  $F : \mathbf{Pol}_f \rightarrow \mathbf{Pol}$  admits a right Kan extension  $\text{Ran}_{I_{cz}} F$  along  $I_{cz}$ . Following (Mac Lane [15], X.3, Corollary 4) it is sufficient to prove that for all  $X \in \text{Obj}(\mathbf{Pol}_{cz})$ , the diagram  $F \circ D_X : (X \downarrow I_{cz}) \rightarrow \mathbf{Pol}$  has a limit. By a cofinality argument similar to that used in the proof of Prop. 3.2, one can show

that  $\lim F \circ D_X \cong \lim F \circ D$  for a countable diagram  $D$  and since  $\mathbf{Pol}$  is countably complete this limit exists, therefore  $F$  admits a right Kan extension. Let us prove that the extension is full and faithful. Since  $I_{cz}$  is full and faithful, the universal arrow  $\epsilon_F : (\text{Ran}_{I_{cz}} F) I_{cz} \Rightarrow F$  is an iso. Given  $F, G : \mathbf{Pol}_f \rightarrow \mathbf{Pol}_{cz}$  and  $\alpha : F \Rightarrow G$ , there exists a unique  $\sigma : \text{Ran}_{I_{cz}} F \Rightarrow \text{Ran}_{I_{cz}} G$  such that  $\alpha \circ \epsilon_F : (\text{Ran}_{I_{cz}} F) I_{cz} \rightarrow G$  factors as  $\alpha \circ \epsilon_F = \epsilon_G \circ \sigma I_{cz}$ . Therefore,  $\text{Ran}_{I_{cz}}$  defines a functor from  $[\mathbf{Pol}_f; \mathbf{Pol}]$  to  $[\mathbf{Pol}_{cz}; \mathbf{Pol}]$  which is full and faithful by the bijection  $\text{Nat}(\text{Ran}_{I_{cz}} F, \text{Ran}_{I_{cz}} G) \cong \text{Nat}(F, G)$ .  $\square$

**Proof.** We use Prop. 4.3 to prove Theorem 4.2. The universal property of  $\text{Ran}$  yields the isomorphism  $\text{Nat}(F|_{\mathbf{Pol}_f}, G|_{\mathbf{Pol}_f}) \cong \text{Nat}(F, \text{Ran}_{I_{cz}} G|_{\mathbf{Pol}_f})$ . Recall that  $\text{Ran}_{I_{cz}} G|_{\mathbf{Pol}_f}(X) = \lim G \circ D_X \cong \lim G \circ D$  where  $D_X$  and  $D$  are as in the proof of Prop. 4.3. By  $\mathbf{Pol}_f$ -continuity of  $G$ ,  $\text{Ran}_{I_{cz}} G|_{\mathbf{Pol}_f}(X) \cong G(\lim D) = G(X)$ .  $\square$

**II. From  $\mathbf{Pol}_{cz}$  to  $\mathbf{Pol}_z^b$ .** As seen in Prop. 3.6, the Wallman compactification makes  $\mathbf{Pol}_{cz}$  into a reflective subcategory of  $\mathbf{Pol}_z^b$ . The extension of a natural transformation from  $\mathbf{Pol}_{cz}$  to  $\mathbf{Pol}_z^b$  can be framed componentwise as a *restriction* of the natural transformation to a space embedded into its compactification, that we construct using intersections.

**Definition 4.4 (Intersections, preservation of intersections)** If  $j_1 : X \hookrightarrow Z, j_2 : Y \hookrightarrow Z$  are two embeddings, we define the intersection  $X \cap Y \rightarrow Z$  as the pullback of  $j_1$  and  $j_2$  (Eq. 5). We say that an endofunctor  $F : \mathbf{Pol} \rightarrow \mathbf{Pol}$  preserves intersections if the diagram in Eq. 6 is an intersection.

$$\begin{array}{ccc} X \cap Y & \xrightarrow{p_1} & X \\ p_2 \downarrow \lrcorner & & \downarrow j_1 \\ Y & \xrightarrow{j_2} & Z \end{array} \quad (5) \qquad \begin{array}{ccc} G(X \cap Y) & \xrightarrow{G(p_1)} & G(X) \\ G(p_2) \downarrow \lrcorner & & \downarrow G(j_1) \\ G(Y) & \xrightarrow{G(j_2)} & G(Z) \end{array} \quad (6)$$

The following Lemma characterises the topology of intersections in  $\mathbf{Pol}$ .

**Lemma 4.5**  $X \cap Y$  is the **Set**-theoretic intersection of  $X, Y$  together with the subspace topology induced by  $Z$ .

**Proof.** The proof is routine.  $\square$

Recall that if  $f : X \rightarrow Y$  is a morphism in a category  $\mathbb{C}$ , its *cokernel pair* (if it exists) is the pushout of  $f$  with itself (Mac Lane [15], III.3). In  $\mathbf{Top}$ , there is a well-known characterisation of embeddings as limits of their cokernel pair (see e.g. (Adamek et al. [1], 7.56-7.58)). In  $\mathbf{Pol}$ , we have the following:

**Proposition 4.6** Let  $f : X \hookrightarrow Y$  be an embedding. Then (i) the pushout object  $Y +_X Y$  is Polish, (ii) the cokernel arrows  $j_1, j_2 : Y \rightarrow Y +_X Y$  are embeddings and (iii) the intersection of  $j_1$  and  $j_2$  is homeomorphic to  $X$ .

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ f \downarrow \lrcorner & & \downarrow j_1 \\ Y & \xrightarrow{j_2} & Y +_X Y \end{array}$$

**Proof.** The proof is routine.  $\square$

The following Lemma ensures that the pushout object of an embedding with range in  $\mathbf{Pol}_{cz}$  is still compact zero-dimensional.



**III. From  $\mathbf{Pol}_z^b$  to  $\mathbf{Pol}$ .** The last part of the Machine is a procedure to extend natural transformations from  $\mathbf{Pol}_z^b$  to  $\mathbf{Pol}$ . We have seen in Prop. 3.11 that Polish spaces are the colimits of their “diagrams of zero-dimensionals”. We will require functors in the domain of natural transformations to commute with these colimits.

**Definition 4.9 (Z-cocontinuous functors)** A functor  $F : \mathbf{Pol} \rightarrow \mathbf{Pol}$  is *Z-cocontinuous* if for all  $X \in \text{Obj}(\mathbf{Pol})$ ,  $F(X) \cong \text{colim } FZ_X$  where  $Z_X$  is defined in Def. 3.10.

Moreover, we will require these functors to be *Z-stable*, which means that the underlying sets of the spaces in the range of the considered functors are invariant by zero-dimensionalisation. As we will prove later, this is for instance the case of the Giry, multiset and list functors.

**Definition 4.10 (Z-stable functor)** A functor  $F : \mathbf{Pol} \rightarrow \mathbf{Pol}$  is *Z-stable* if  $UF_X = UFZ_X(\mathcal{F})$  for all  $\mathcal{F} \in \text{Bases}(X)$ .

**Theorem 4.11** Let  $F, G : \mathbf{Pol} \rightarrow \mathbf{Pol}$  be a pair of functors such that  $F$  is Z-cocontinuous and Z-stable. Then  $\text{Nat}(F, G) \cong \text{Nat}(FU_p I_p^b, GU_p I_p^b)$ .

**Proof.** Let  $\alpha : FU_p I_p^b \Rightarrow GU_p I_p^b$  and  $X \in \text{Obj}(\mathbf{Pol})$  be given. By Z-cocontinuity,  $F(X)$  is the colimiting object of the diagram  $FZ_X = FU_p I_p^b z B_X : \text{Bases}(X)^{op} \rightarrow \mathbf{Pol}$  (Def. 3.10). Applying  $\alpha$ , we get a natural transformation  $\alpha z B_X : FZ_X \Rightarrow GZ_X$ . Composing with the counit  $\epsilon : I_p^b z \rightarrow Id_{\mathbf{Pol}^b}$  yields a natural transformation  $(GU_p \epsilon)(\alpha z B_X) : FZ_X \Rightarrow GU_p Id_{\mathbf{Pol}^b} B_X$ . Note that  $GU_p Id_{\mathbf{Pol}^b} B_X$  is equal to the constant functor with value  $G(X)$ . Therefore, we have constructed a cocone from  $FZ_X$  to  $G(X)$ . The situation above is summed up in the following diagram:

$$\begin{array}{ccccccc}
 & & & & \mathbf{Pol}^b & \xrightarrow{U_p} & \mathbf{Pol} \\
 & & & & \uparrow I_p^b & & \uparrow F \\
 \text{Bases}(X)^{op} \xrightarrow{B_X} & \mathbf{Pol}^b & \xrightarrow{z} & \mathbf{Pol}_z^b & & & \\
 & \downarrow \epsilon & & \downarrow I_p^b & & & \\
 & & & \mathbf{Pol}^b & \xrightarrow{U_p} & \mathbf{Pol} & \\
 & & \text{Id}_{\mathbf{Pol}^b} & & & & \uparrow G
 \end{array}$$

By universality, there exists a unique map  $u_X : F(X) \rightarrow G(X)$  such that  $u_X \circ (FU_p \epsilon B_X)_{\mathcal{F}} = (GU_p \epsilon B_X)_{\mathcal{F}} \circ (\alpha z B_X)_{\mathcal{F}}$ . Let us prove naturality of  $\{u_X\}_{X \in \text{Obj}(\mathbf{Pol})}$ . For all  $f : X \rightarrow Y$  and for all base  $\mathcal{G}$  of  $Y$ , there exists a base  $\mathcal{F}$  of  $X$  such that  $f : (X, \mathcal{F}) \rightarrow (Y, \mathcal{G})$ , is base-preserving, and by functoriality, so is  $z(f) : Z_X(\mathcal{F}) \rightarrow Z_Y(\mathcal{G})$ . We get the following diagram:

$$\begin{array}{ccccc}
 FZ_X(\mathcal{F}) & \xrightarrow{(\alpha z B_X)_{\mathcal{F}}} & GZ_X(\mathcal{F}) & & \\
 \downarrow Fz(f) & \searrow (FU_p \epsilon B_X)_{\mathcal{F}} & \downarrow (GU_p \epsilon B_X)_{\mathcal{F}} & & \\
 & FX & \xrightarrow{u_X} & GX & \\
 & \downarrow F(f) & & \downarrow G(f) & \\
 & FY & \xrightarrow{u_Y} & GY & \\
 & \uparrow & & \uparrow & \\
 FZ_Y(\mathcal{G}) & \xrightarrow{(\alpha z B_Y)_{\mathcal{G}}} & GZ_Y(\mathcal{G}) & & \\
 & \downarrow Fz(f) & & \downarrow Gz(f) & 
 \end{array}$$

In the above diagram, the left and right cells commute by naturality of  $\epsilon$  while the top and bottom cells commute by construction of the arrows  $u_X, u_Y$ . Note that the arrow  $(FU_p \epsilon B_X)_{\mathcal{F}}$  is the image through  $F$  of the identity function  $\epsilon_{\mathcal{F}} = id : Z_X(\mathcal{F}) \rightarrow X$ . Since  $F$  is  $Z$ -stable, this arrow is surjective. We conclude that the central square commute, and we extend  $\alpha$  by setting for all  $X$   $\alpha_X = u_X$  as constructed above.  $\square$

**IV. The Machine.** Bringing the parts of the Machine together, we obtain:

**Theorem 4.12** *Let  $F, G : \mathbf{Pol} \rightarrow \mathbf{Pol}$  be a pair of functors such that:*

- (i)  *$F$  is  $Z$ -cocontinuous and  $Z$ -stable,*
- (ii)  *$G$  is  $\mathbf{Pol}_f$ -continuous, preserves embeddings and intersections.*

*Then one has  $\text{Nat}(F, G) \cong \text{Nat}(F|_{\mathbf{Pol}_f}, G|_{\mathbf{Pol}_f})$ .*

## 5 Feeding the Machine

We now investigate the properties of some functors, with an eye on applying the Machine.

**The Giry functor.** For any space  $X$ , we denote by  $\mathbf{G}(X)$  the space of Borel probability measures over  $X$ , endowed with the weak topology (Giry, [9]). This operation can be extended to a functor  $\mathbf{G} : \mathbf{Pol} \rightarrow \mathbf{Pol}$  which admits the Giry monad structure  $(G, \delta, \mu)$  (Giry, [9]). The action of  $\mathbf{G}$  on maps  $f : X \rightarrow Y$  is defined by  $\mathbf{G}(f)(P) \triangleq P \circ f^{-1}$ . The unit is given by the Dirac delta:  $\delta_X : X \rightarrow \mathbf{G}(X)$  while the multiplication is defined by averaging:  $\mu_X : \mathbf{G}^2(X) \rightarrow \mathbf{G}(X) \triangleq P \mapsto \int_{\mathbf{G}(X)} p dP(p)$ .  $\mathbf{G}$  is a rather well-behaved functor:

**Proposition 5.1** *(i) For all ccd  $D$ ,  $\mathbf{G}(\lim D) \cong \lim \mathbf{G} \circ D$ ; (ii)  $\mathbf{G}$  is  $Z$ -cocontinuous and  $Z$ -stable; (iii)  $\mathbf{G}$  preserves injections and embeddings; (iv)  $\mathbf{G}$  preserves intersections.*

**Proof.** (i) is the Bochner extension theorem in functorial form ([7], Theorem 2.5). (ii)  $Z$ -cocontinuity is in ([7], Theorem 3.7);  $Z$ -stability stems from Prop. 3.7, 3. For (iii), see e.g. ([7], Lemma 2.1). Now for (iv): let  $j_1, j_2 : A, B \hookrightarrow X$  be two embeddings, let  $p_1 : A \cap B \rightarrow A$  and  $p_2 : A \cap B \rightarrow B$  be the corresponding embeddings and consider  $\mu \in \mathbf{G}(A), \nu \in \mathbf{G}(B)$  such that  $\mathbf{G}(j_1)(\mu) = \mathbf{G}(j_2)(\nu)$ . It follows from (Kechris [11], Theorem 15.1) and the fact that  $p_1$  is injective that whenever  $U$  is a Borel set of  $A \cap B$ ,  $p_1[U]$  is a Borel set of  $A$ , and similarly for  $p_2$ . We can therefore define  $\lambda \in \mathbf{G}(A \cap B)$  by  $\lambda(U) = \mu(p_1[U]) = \nu(p_2[U])$ . To see that the equality on the right holds, note that since  $j_1$  is injective  $p_1[U] = j_1^{-1}(j_1[p_1[U]])$ , and thus

$$\begin{aligned} \mu(p_1[U]) &= \mu(j_1^{-1}(j_1[p_1[U]])) = \mathbf{G}(j_1)(\mu)(j_1[p_1[U]]) = \mathbf{G}(j_2)(\nu)(j_1[p_1[U]]) \\ &= \mathbf{G}(j_2)(\nu)(j_2[p_2[U]]) = \nu(p_2[U]) \end{aligned}$$

This assignment from pairs  $(\mu, \nu)$  such that  $\mathbf{G}j_1(\mu) = \mathbf{G}j_2(\nu)$  to  $\lambda \in \mathbf{G}(A \cap B)$  is clearly injective, and it follows that  $\mathbf{G}(A \cap B) \cong \mathbf{G}A \cap \mathbf{G}B$  as sets. Since  $\mathbf{G}$  preserves embeddings,  $\mathbf{G}(j_1 \circ p_1) = \mathbf{G}(j_2 \circ p_2)$  is an embedding, and it follows that  $\mathbf{G}(A \cap B)$  and  $\mathbf{G}A \cap \mathbf{G}B$  are in fact homeomorphic.  $\square$

**Example 5.2** Theorem 4.12 implies that the monadic data of the Giry monad is entirely determined on  $\mathbf{Pol}_z$  by its finite components. We conjecture this holds for arbitrary Polish spaces.

**The non-zero finite measures functors.** We will also consider functors closely related to  $\mathbf{G}$ : we let  $\mathbf{M}^+$  be the functor mapping any space  $X$  to the space of non-zero positive finite measures over  $X$  with the weak topology, and acting on maps similarly as  $\mathbf{G}$ . The functor of non-zero signed finite measures over  $X$ , denoted by  $\mathbf{M}^*$ , is defined similarly. See ([7], Sec. 2) for more details. The following is trivial (consider the normalisation of a finite non-zero measure):

**Proposition 5.3** *For all space  $X$ , we have the isomorphism  $\mathbf{M}^+(X) \cong \mathbf{G}(X) \times \mathbb{R}_{>0}$  and  $\mathbf{M}^*(X) \cong \mathbf{G}(X) \times \mathbb{R}^*$ , where  $\mathbb{R}^* = \mathbb{R} \setminus \{0\}$ .*

As a consequence,  $\mathbf{M}^+$  and  $\mathbf{M}^*$  verify all the properties listed in Prop. 5.1. Note that for all finite space  $n$ ,  $\mathbf{M}^+(n)$  is also homeomorphic to  $\mathbb{R}_{\geq 0}^n \setminus \{0\}$ .

**The multiset functor.** We consider the multiset functor  $\mathbf{B} : \mathbf{Pol} \rightarrow \mathbf{Pol}$ . It is given explicitly by

$$\mathbf{B}(X) \triangleq \coprod_{n \in \mathbb{N}} X^n / S_n$$

where  $X^n / S_n$  is the quotient of  $X^n$  under the obvious action of  $S_n$  on tuples with the quotient topology, i.e. the final topology for the quotient map  $q : X^n \twoheadrightarrow X^n / S_n$ . See Appendix A for a proof that  $\mathbf{B}(X)$  is Polish. Its action on maps is given by setting for any  $f : X \rightarrow Y$  and  $\mu \in \mathbf{B}(X)$ ,  $\mathbf{B}(f)(\mu) = y \mapsto \sum_{x \in f^{-1}(y)} \mu(x)$ . This is easily shown to be continuous. Observe also that for  $X$  finite,  $\mathbf{B}(X) \cong \mathbb{N}^X$ . The multiset functor verifies the following properties:

**Proposition 5.4** *(i)  $\mathbf{B}$  is  $\mathbf{Pol}_f$ -continuous; (ii)  $\mathbf{B}$  preserves injections and embeddings; (iii)  $\mathbf{B}$  preserves intersections.*

**Proof.** See Appendix B. □

**The Vietoris functor.** As a non-probabilistic example, we will consider the Vietoris functor. We recall its definition.

**Definition 5.5** We denote by  $\mathbf{V} : \mathbf{Pol} \rightarrow \mathbf{Pol}$  the functor mapping any space  $X$  to the space of compact subsets of  $X$  topologised with the Hausdorff distance, and mapping any continuous function  $f : X \rightarrow Y$  to  $\mathbf{V}(f) \triangleq K \in \mathbf{V}(X) \mapsto f(K)$ .

See (Kechris [11], 4.F) for a proof that  $\mathbf{V}(X)$  is indeed Polish.  $\mathbf{V}$  has the following properties:

**Proposition 5.6** *(i)  $\mathbf{V}$  is  $\mathbf{Pol}_f$ -continuous; (ii)  $\mathbf{V}$  preserves injections and embeddings; (iii)  $\mathbf{V}$  preserves intersections.*

**Proof.** (i) is in Appendix B. (ii) and (iii) are in Appendix B. □

**Example 5.7** An interesting example is provided by the *support* of a measure. Usually, the support of  $p \in \mathbf{G}(X)$  is defined to be the smallest closed subset of measure 1. On finite spaces, for  $p \in \mathbf{G}(n)$ , we define  $\text{supp}_n(p) \triangleq \{x \in n \mid p(x) > 0\}$ .

Let us check that this is natural: for  $f : m \rightarrow n$ , we have that  $\text{supp}(G(f)(p)) = \text{supp}(p \circ f^{-1}) = \{x \in n \mid f^{-1}(x) \cap \text{supp}(p) \neq \emptyset\}$ , i.e.  $\text{supp}(G(f)(p)) = f(\text{supp}(p)) = V(f)(\text{supp}(p))$ . Therefore,  $\text{supp} : \mathbf{G}|_{\mathbf{Pol}_f} \Rightarrow \mathbf{V}|_{\mathbf{Pol}_f}$  is a natural transformation. The Machine (Theorem 4.12) uniquely extends  $\text{supp}$  to a natural transformation  $\text{supp} : \mathbf{G} \Rightarrow \mathbf{V}$ . This type is rather unusual, as the support of a probability measure is closed but not generally compact.

## 6 Rigidity

The results presented in Sec. 4 allow to construct natural transformations from finitary specifications. In this section, we apply these results to exhibit striking *rigidity* properties of  $\mathbf{G}$  and related functors.

**Definition 6.1** A pair of functors  $F, G : \mathbb{C} \rightarrow \mathbb{D}$  is called *rigid*, if there exists at most one natural transformation  $\eta : F \Rightarrow G$ . In particular, we will say that an endofunctor  $F : \mathbb{C} \rightarrow \mathbb{D}$  is **rigid** if the identity natural transformation  $\text{id} : F \Rightarrow F$  is the only natural transformation that exists from  $F$  to itself.

For each finite space  $k$  and functor  $T : \mathbf{Pol} \rightarrow \mathbf{Pol}$ , there exists a canonical action of  $S_k$ , the permutation group over  $k$  elements, given by:

$$\alpha : S_k \times T(k) \rightarrow T(k), (\pi, x) \mapsto T(\pi(x))$$

We will call this action the *canonical action*. We will call an element  $x \in T(k)$  stabilised by the entire group  $S_k$  under the canonical action an *isotropic element*. Isotropic elements will play a crucial role in our theorem.

**Theorem 6.2 (Rigidity Theorem)** *Let  $H : \mathbf{Pol} \rightarrow \mathbf{Pol}$  be a subfunctor of the Giry monad  $\mathbf{G}$  satisfying the following conditions: (i)  $H(k) = \mathbf{G}(k)$  for every finite Polish space  $k$ ; (ii)  $H$  is  $\mathbf{Pol}_f$ -continuous; (iii)  $H$  preserves injections. Let also  $T : \mathbf{Pol} \rightarrow \mathbf{Pol}$  be a functor such that (iv) for each finite Polish space  $k$  there exists a dense subset  $Q_k \subseteq T(k)$  with the property that if  $x \in Q_k$  there exists a finite Polish space  $k'$ , a morphism  $f : k' \rightarrow k$  and an isotropic element  $x' \in T(k')$  such that  $T(f)(x') = x$ . In these circumstances the pair  $(T, H)$  is rigid.*

We prove this theorem in steps. But let us first show some example of functors satisfying the property above.

**Example 6.3** Let us show that the Vietoris functor  $\mathbf{V}$  satisfies the condition (iv). Note first that for every  $k$ , the full set  $k \in \mathbf{V}(k)$  is isotropic: for any  $\pi \in S_k$   $\alpha(\pi, k) = \mathbf{V}\pi(k) = k$  since  $\pi$  is bijective. Now take  $Q_k = \mathbf{V}(k)$  (which is trivially dense) and  $x = \{x_1, \dots, x_n\} \in \mathbf{V}(k)$ . Consider the full set  $n \in \mathbf{V}(n)$  along with the map  $f : n \rightarrow k, i \mapsto x_i$ , it is clear that  $\mathbf{V}(f)(n) = x$ , and  $n$  is isotropic.

**Example 6.4** The Giry monad  $\mathbf{G}$  satisfies all conditions of Theorem 6.2: it satisfies (i) trivially, it satisfies (ii) and (iii) by Prop. 5.1. Let us show that it satisfies (iv) as well. Note first that the uniform probabilities are the isotropic elements: if  $(\frac{1}{k}, \dots, \frac{1}{k})$  denotes the uniform distribution on  $k$  elements, then

$$\alpha\left(\pi, \left(\frac{1}{k}, \dots, \frac{1}{k}\right)\right) = \mathbf{G}(\pi)\left(\frac{1}{k}, \dots, \frac{1}{k}\right) = \left(\frac{1}{k}, \dots, \frac{1}{k}\right) \circ \pi^{-1} = \left(\frac{1}{k}, \dots, \frac{1}{k}\right)$$



Consider now  $Q_k = \Delta_k \cap \mathbb{Q}^k$ , the rational probabilities on  $k$  elements. It is clearly dense in  $\mathbf{G}(k)$ . Any  $x \in Q_k$ , can without loss of generality be written as  $(\frac{p_1}{n}, \dots, \frac{p_m}{n})$  for a common denominator  $n$ . Now consider the projection map defined by

$$p : n \rightarrow k, i \mapsto \begin{cases} 1 & \text{if } 1 \leq i \leq p_1 \\ 2 & \text{if } p_1 + 1 \leq i \leq p_1 + p_2 \\ \dots & \\ k & \text{if } \sum_{i=1}^{k-1} p_i + 1 \leq i \leq \sum_{i=1}^k p_i \end{cases}$$

It is easy to check from this definition that  $(\frac{p_1}{n}, \dots, \frac{p_m}{n}) = \mathbf{G}(p)(\frac{1}{n}, \dots, \frac{1}{n})$ , where  $(\frac{1}{n}, \dots, \frac{1}{n})$  is isotropic.

**Example 6.5** Let  $\mathbf{M}^+, \mathbf{M}^* : \mathbf{Pol} \rightarrow \mathbf{Pol}$  be the finite non-zero positive (resp. signed) measure functors, then  $\mathbf{M}^+(k) \cong G(k) \times \mathbb{R}_{>0}$  and  $\mathbf{M}^*(k) \cong \mathbf{G}(k) \times \mathbb{R}^*$ . These functors satisfy condition (iv): the isotropic elements are those of the shape  $((1/k, \dots, 1/k), \lambda)$  for  $\lambda \in \mathbb{R}^*$  or  $\mathbb{R}_{>0}$ . A dense subset is provided by  $(Q^k \cap \mathbf{G}(k)) \times \mathbb{R}^*$  and  $(Q^k \cap G(k)) \times \mathbb{R}_{>0}$  respectively and the same argument as in Example 6.4 shows that every element  $((p_1/n, \dots, p_k/n), \lambda)$  is the image of  $((1/n, \dots, 1/n), \lambda)$  by  $\mathbf{G}(p) \times id$  with  $p$  defined as in Example 6.4.

**Example 6.6** The multiset functor  $\mathbf{B}$  also has the property (iv).  $\mathbf{B}(k)$  has one isotropic element: the unordered list  $[(1, \dots, k)]$ , and any  $[(x_1, \dots, x_k)] \in \mathbf{B}(k)$  is the image of  $[(1, \dots, k)]$  under  $\mathbf{B}(f)$  for the map  $f : k \rightarrow k, i \mapsto x_i$  (which might very well not be injective).

Let us proceed to the proof of Theorem 6.2. The following settles the finite case:

**Lemma 6.7** *Let  $(T, H)$  be a pair of functors satisfying the conditions of Theorem 6.2, then  $(T, H)$  is rigid on  $\mathbf{Pol}_f$ .*

**Proof.** Let  $\nu : T \Rightarrow H$  be a natural transformation. We first show that if  $x \in T(k)$  is isotropic then

$$\nu_k(x) = \left( \frac{1}{k}, \dots, \frac{1}{k} \right) \quad (7)$$

where  $(\frac{1}{k}, \dots, \frac{1}{k})$  denotes the uniform probability distribution on  $k$ . Fix  $i \in \{1, \dots, k\}$ , and consider the permutations  $(ij) \in S_k, 1 \leq j \leq k$  sending  $i$  to  $j$ ,  $j$  to  $i$  and leaving all other elements of  $k$  unchanged. We have

$$\begin{aligned} \nu_k(x)(i) &= \nu_k(T(ij)(x))(i) && (x \text{ isotropic}) \\ &= H(ij)(\nu_k(x))(i) && (\text{By naturality}) \\ &= \mathbf{G}(ij)(\nu_k(x))(i) && (H = \mathbf{G} \text{ on } \mathbf{Pol}_f) \\ &= \nu_k(x)(ij)^{-1}(i) && (\text{By def. of } \mathbf{G}) \\ &= \nu_k(x)(j) && (\text{By def. of } (ij)) \end{aligned}$$

Since this holds for every  $1 \leq j \leq k$  we have  $\sum_{j=1}^k \nu_k(x)(j) = \sum_{j=1}^k \nu_k(x)(i) = k\nu_k(x)(i) = 1$  and thus  $\nu_k(x)(i) = \frac{1}{k}$  for every  $1 \leq i \leq k$ , i.e.  $\nu_k(x) = (\frac{1}{k}, \dots, \frac{1}{k})$ .



Let us now consider an arbitrary  $x \in Q_k$ , by assumption there exist  $f : k' \rightarrow k$  and an isotropic element  $x' \in T(k')$  such that  $T(f)(x') = x$ . It follows that

$$\begin{aligned}
\nu_k(x) &= \nu_k(T(f)(x')) && \text{(By assumption on } T) \\
&= H(f)(\nu_k(x')) && \text{(By naturality)} \\
&= G(f)(\nu_k(x')) && (H = G \text{ on } \mathbf{Pol}_f) \\
&= G(f)\left(\frac{1}{k'}, \dots, \frac{1}{k'}\right) && (x' \text{ is isotropic and (7)})
\end{aligned}$$

Clearly, the same reasoning applies to any other natural transformation  $\rho : T \Rightarrow H$ . We have thus shown that for each finite Polish set  $k$ ,  $\nu_k$  is unique on a dense subset  $Q_k$  of  $T(k)$ . Since  $\nu_k$  is a morphism in  $\mathbf{Pol}$  it is continuous, and since Polish spaces are complete, it is in fact Cauchy-continuous. It follows that the restriction of  $\nu_k$  to  $Q_k$  has a unique extension to  $T(k)$ . Since the restriction of  $\nu_k$  to  $Q_k$  is unique, it follows that  $\nu_k$  is also unique.  $\square$

Note that the entire group  $S_k$  was necessary to show Lemma 6.7, i.e. a weaker notion of isotropic element would not be sufficient.

**Lemma 6.8** *Let  $(T, H)$  be a pair of functors satisfying the conditions of Theorem 6.2, then  $(T, H)$  is rigid on  $\mathbf{Pol}_{cz}$ .*

**Proof.** Assume  $\nu : T|_{\mathbf{Pol}_f} \Rightarrow H|_{\mathbf{Pol}_f}$  is given. By Lemma 6.7,  $\nu$  is unique. Since  $H$  is  $\mathbf{Pol}_f$ -continuous, Theorem 4.2 applies and the proof is complete.  $\square$

**Lemma 6.9** *Let  $(T, H)$  be a pair of functors satisfying the conditions of Theorem 6.2, then  $(T, H)$  is rigid on  $\mathbf{Pol}_z^b$ .*

**Proof.** It is enough to reuse the uniqueness part of the proof of Theorem 4.8.  $\square$

We can finally prove Theorem 6.2.

**Proof. (Theorem 6.2)** Let  $\alpha : T|_{\mathbf{Pol}_z^b} \Rightarrow H|_{\mathbf{Pol}_z^b}$  be given. By Lemma 6.9,  $\alpha$  is the unique such transformation. Let  $\beta, \beta' : T \Rightarrow H$  be given, extending  $\alpha$ . For all  $X$  and  $\mathcal{F} \in \text{Bases}(X)$ , the identity function  $id : z_{\mathcal{F}}(X) \rightarrow X$  is continuous. By the rigidity assumption,  $\beta_{z_{\mathcal{F}}(X)} = \beta'_{z_{\mathcal{F}}(X)}$ . Using this equation and naturality,

$$\beta_X \circ T(id) = H(id) \circ \beta_{z_{\mathcal{F}}(X)} = H(id) \circ \beta'_{z_{\mathcal{F}}(X)} = \beta'_X \circ T(id)$$

Therefore  $\beta = \beta'$ .  $\square$

**Example 6.10** We have shown earlier that  $G$  satisfies all the conditions of Theorem 6.2. It follows that there can only exist a single natural transformation  $G \Rightarrow G$ , and since the identity transformation is natural, it follows that  $G$  is rigid.

**Example 6.11** Let  $M^+ : \mathbf{Pol} \rightarrow \mathbf{Pol}$  be the finite positive measure functor. We can check that the following transformation is natural: define  $\nu : M^+ \rightarrow G$  at a Polish space  $X$  by  $\nu_X(Q) \triangleq A \mapsto \frac{Q(A)}{Q(X)}$  for  $A$  a Borel set of  $X$ . This is well defined since  $0 < Q(X) < \infty$ . It is also natural: if  $f : X \rightarrow Y$  is a map in  $\mathbf{Pol}$ , then for each  $Q$

in  $\mathbf{M}^+(X)$  and Borel set  $B$  of  $Y$  we have:

$$\begin{aligned} \mathbf{G}(f)(\nu_X(Q))(B) &= \nu_X(Q)(f^{-1}(B)) = \frac{Q(f^{-1}(B))}{Q(X)} = \frac{Q(f^{-1}(B))}{Q(f^{-1}(Y))} \\ &= \nu_Y(\mathbf{M}^+(f)(Q))(B) \end{aligned}$$

Since  $\mathbf{M}^+$  satisfies (iv), it follows from Theorem 6.2, that the normalisation transformation  $\nu$  we have just defined is the only natural transformation  $\mathbf{M}^+ \Rightarrow \mathbf{G}$ .

## 7 Applications

In previous work [7], we showed that a cornerstone of nonparametric Bayesian statistics, the Dirichlet process [8,10], is in fact a natural transformation from  $\mathbf{M}^+$  to  $\mathbf{G}^2$ . This result hinged on a non-axiomatic version of the Machine of Sec. 4. In order to validate our new developments we first give a short construction of the Dirichlet process in axiomatic form. The value of our general framework is then illustrated by constructing the Poisson process as a natural transformation. At the heart of these constructions are families of distributions which are stable by convolution (mistakenly taken to be infinitely divisible in [7]). Common examples include: the  $\Gamma$  distribution, the Gaussian distribution, the Poisson distribution, etc. What examples such as Dirichlet or Poisson processes have in common is that they can all be represented by natural transformations of the shape  $\mathbf{M}^+ \Rightarrow \mathbf{G}H$  where the functor  $H$  can be either  $\mathbf{B}$  or  $\mathbf{M}^+$ . Since  $\mathbf{M}^+$  is  $Z$ -cocontinuous, since  $\mathbf{G}$  and  $H$  are  $\mathbf{Pol}_f$ -continuous, preserve injections, embeddings and intersections (see Appendix B) we can define a natural transformation of this type by restricting ourselves to  $\mathbf{Pol}_f$  and running the Machine.

In the cases which we have mentioned above, the natural transformation in  $\mathbf{Pol}_f$  can in fact be defined by a single map! The fundamental property which makes this possible is that both  $\mathbf{M}^+$  and  $\mathbf{B}$  turn coproducts into products. When this is the case it is sometimes possible to define  $\phi : \mathbf{M}^+ \Rightarrow \mathbf{G}H$  on  $\mathbf{Pol}_f$  from a map  $\phi_1 : \mathbf{M}^+(1) \rightarrow \mathbf{G}H(1)$ . For this we need a fundamental result which holds very generally in the category  $\mathbf{Meas}$  of measurable spaces and measurable maps. We define the *product measure* natural transformation between the bifunctors  $\pi : \mathbf{G} - \times \mathbf{G} - \rightarrow \mathbf{G}(- \times -)$  at each pair of measurable spaces  $((X, \Sigma_X), (Y, \Sigma_Y))$  by  $\pi_{(X,Y)}(p, q) \mapsto p \times q$  where  $p \times q$  is the product measure defined on the product  $\sigma$ -algebra  $(\Sigma_X \otimes \Sigma_Y)$ .

**Theorem 7.1** *The transformation  $\pi : \mathbf{G} - \times \mathbf{G} - \rightarrow \mathbf{G}(- \times -)$  is natural in both its arguments.*

**Proof.** The proof is routine. □

Let us now fix a continuous map  $\phi_1 : \mathbf{M}^+(1) \rightarrow \mathbf{G}H(1)$ . For any  $n$  in  $\mathbf{Pol}_f$  we use the fact that  $n = \coprod_{i=1}^n 1$  and the fact that  $\mathbf{M}^+$  and  $H$  turn coproducts into products to define  $\phi_n : \mathbf{M}^+(n) \rightarrow \mathbf{G}H(n)$  by

$$\mathbf{M}^+(n) \cong \mathbf{M}^+(1)^n \xrightarrow{\phi_1^n} (\mathbf{G}H(1))^n \xrightarrow{\otimes_{H1}} \mathbf{G}(H1)^n \cong \mathbf{G}H(n)$$

where  $\bigotimes_{H(1)}$  is the  $n$ -fold measure product at  $H(1)$ . The maps  $\phi_n$  define the component of a transformation  $M^+ \Rightarrow GH$ . But when is it natural? A simple criterion is given in the following result.

**Theorem 7.2** *A transformation  $\phi : M^+ \rightarrow GH$  built as above is natural in  $\mathbf{Pol}_f$  iff the following diagrams commute:*

$$\begin{array}{ccc} M^+(2) & \xrightarrow{\phi_2} & GH(2) \\ M^+(e) \downarrow & & \downarrow GH(e) \\ M^+(1) & \xrightarrow{\phi_1} & GH(1) \end{array} \quad (8)$$

$$\begin{array}{ccc} M^+(n) & \xrightarrow{\phi_n} & GH(n) \\ M^+(ij) \downarrow & & \downarrow GH(ij) \\ M^+(n) & \xrightarrow{\phi_n} & GH(n) \end{array} \quad (9)$$

$$\begin{array}{ccc} M^+(1) & \xrightarrow{\phi_1} & GH(1) \\ M^+(i_1) \downarrow & & \downarrow GH(i_1) \\ M^+(2) & \xrightarrow{\phi_2} & GH(2) \end{array} \quad (10)$$

$$\begin{array}{ccc} M^+(1) & \xrightarrow{\phi_2} & GH(2) \\ M^+(i_2) \downarrow & & \downarrow GH(i_2) \\ M^+(2) & \xrightarrow{\phi_1} & GH(2) \end{array} \quad (11)$$

where  $e : 2 \rightarrow 1$  is the obvious unique epimorphism,  $(ij) : n \rightarrow n$  is any permutation of two elements of  $n$ , and  $i_1, i_2 : 1 \rightarrow 2$  are the two injections of 1 into  $2 = 1 + 1$ .

**Proof.** Any map  $f : m \rightarrow n$  between finite sets can be written as a permutation  $\pi : n \rightarrow n$  followed by a monotone surjection  $q : n \rightarrow k$  followed by a monotone injection  $i : k \rightarrow n$ . Since every permutation of  $n$  can be written as a composition of permutation of two elements, repeated usage of Diagram (9) shows that  $GH\pi \circ \phi_n = \phi_n \circ M^+\pi$ . Monotone surjections  $q : m \rightarrow n$  can be written as a composition of maps of the shape

$$id_1 + id_1 + \dots + e + id_1 + \dots + id : k \rightarrow k - 1$$

For notational clarity let us consider the case  $e + id_1 : 3 \rightarrow 2$ . The following square commutes:

$$\begin{array}{ccccc} M^+(3) \cong M^+(2) \times M^+(1) & \xrightarrow{\phi_2 \times \phi_1} & GH(2) \times GH(1) & \xrightarrow{\bigotimes} & G(H(2) \times H(1)) \cong GH(3) \\ M^+(e) \times id_1 \downarrow & & \downarrow GH(e) \times id_1 & & \downarrow G(H(e) \times id_1) \\ M^+(2) \cong M^+(1) \times M^+(1) & \xrightarrow{\phi_1 \times \phi_1} & GH(1) \times GH(1) & \xrightarrow{\bigotimes} & G(H(1) \times H(1)) \cong GH(2) \end{array}$$

Indeed, the right-hand side square commutes by Theorem 7.1, whilst the left-hand side square commutes by assumption that Diagram 8 commutes. Monotone injections are treated in a similar way.  $\square$

We will call a family of probability distributions  $\phi_n : M^+(n) \rightarrow GH(n)$  *additive* if (8) holds, *exchangeable* if (9) holds, and say that it *admits zero parameters* if (10) and (11) hold.

The  $\Gamma$  distribution  $\Gamma_1 M^+(1) \rightarrow GM^+(1)$  maps any parameter  $\lambda \in M^+(1)$  to a probability with density  $x \mapsto \frac{x^{\lambda-1} e^{-x}}{\Gamma(\lambda)}$  w.r.t. Lebesgue [3]. The family of probability

distributions  $\Gamma_n$  generated by  $\Gamma_1$  is clearly exchangeable; it is also additive [7] and one can easily adapt the definition so that it admits zero parameters. It follows from Theorem 7.2 that  $\Gamma_n : \mathbf{M}^+(n) \rightarrow \mathbf{GM}^+(n)$  is a natural transformation on  $\mathbf{Pol}_f$  which extends to  $\mathbf{Pol}$ . The Dirichlet process is then simply defined as  $\mathcal{D} : \mathbf{M}^+ \Rightarrow G^2 \triangleq (\mathbf{G}\nu)\Gamma$ , where  $\nu : \mathbf{M}^+ \Rightarrow \mathbf{G}$  is the normalisation natural transformation (unique, by rigidity!).

Similarly, if we define  $\Pi_1 : \mathbf{M}^+(1) \rightarrow \mathbf{GB}(1) \cong \mathbf{G}(\mathbb{N})$  by  $\Pi_1(\lambda)(k) = \frac{\lambda^k e^{-\lambda}}{k!}$ , then it is well-known that the family  $\Pi_n$  generated by  $\Pi_1$  (similarly to the previous case) is additive. It is also clearly exchangeable. Finally to allow for zero parameters, we extend  $\Pi_1 : \mathbf{M}_{\geq 0}(1) \rightarrow \mathbf{G}(\mathbb{N})$  by putting  $\Pi_1(0) = \delta_0$ , the Dirac delta at 0. It is clear that for any test function  $f : \mathbb{N} \rightarrow \mathbb{R}$

$$\sum_{k=0} f(k) \frac{\lambda^k e^{-\lambda}}{k!} = f(0)e^{-\lambda} + \sum_{k=1} f(k) \frac{\lambda^k e^{-\lambda}}{k!} \xrightarrow{\lambda \rightarrow 0} f(0) = \sum_k f(k) \delta_0$$

i.e. our extension is continuous for the weak topology. This fact is the exact analogue of Proposition 4.2 in [7]. The family  $\Pi_n : \mathbf{M}_{\geq 0}(n) \rightarrow \mathbf{GN}^n$  thus defines a natural transformation in  $\mathbf{Pol}_f$  by Theorem 7.2, and by applying the Machine we produce a natural transformation on  $\mathbf{Pol}$ . The processes  $\Pi_X : \mathbf{M}_{\geq 0}^+(X) \rightarrow \mathbf{GB}(X)$  (for  $X$  in  $\mathbf{Pol}$ ) defined by this natural transformation are very well-known in probability theory, they are the (inhomogeneous) *Poisson point processes* on  $X$  parameterised by a measure on  $X$ .

## 8 Outlook

Our results allow the compositional and finitary approximation of a class of parameterised “stochastic” processes seen as natural transformations between probability-like functors satisfying some general axioms. It is worth noting that all the conditions on endofunctors that we require for the codomain of natural transformations are preserved by composition (if we strengthen  $\mathbf{Pol}_f$ -continuity to commutation with all limits of *ccds*). Indeed, we are confident that compositionality can be pushed further: following coalgebraic practice, we will investigate whether functors in e.g. the polynomial closure of *Giry* can be fed to the Machine. For this to happen, parts of the Machine have yet to be better understood, in particular the special role played by the requirement of *Z*-cocontinuity (commutation with diagrams of zero-dimensional refinements). For instance, we ignore whether the Vietoris functor and the multiset functors are *Z*-cocontinuous, or whether *Z*-cocontinuity is preserved by composition.

Rigidity is an unexpected mathematical outcome of our structural decomposition of  $\mathbf{Pol}$ . Where the Machine allows to prove existence of natural transformations, rigidity allows to prove *unicity* and is somewhat dual to the former. We expect that the notion of isotropic element will find applications beyond the scope of these developments.

On the applications side, we are confident that many processes beside Dirichlet and Poisson can be subject to the same treatment. Poisson-Dirichlet, Cox processes and some form of Gaussian processes seem to be easy targets. In the case of Dirich-

let, we already know that the Machine allows to prove an asymptotic “learning” property. The work of (Culbertson et al, [6]) will provide a convenient setting where we will study how topological properties of Bayesian models such as continuity relate to asymptotic properties of Bayesian update. The finitary handle provided by the Machine might also be useful in deriving new computability or complexity results in the field of probability.

## References

- [1] Jiri Adamek, Horst Herrlich, and George E. Strecker. *Abstract and concrete categories : the joy of cats*. Pure and applied mathematics. Wiley, New York, 1990. A Wiley-Interscience publication.
- [2] Charalambos D Aliprantis and Kim C Border. *Infinite dimensional analysis*. Springer, 1999.
- [3] N. Balakrishnan and V.B. Nevzorov. *A Primer on Statistical Distributions*. Wiley, 2004.
- [4] N. Bourbaki. *Elements de mathématique. Topologie Générale*. Springer, 1971.
- [5] Philippe Chaput, Vincent Danos, Prakash Panangaden, and Gordon Plotkin. Approximating Markov Processes by averaging. *Journal of the ACM*, 61(1), January 2014. 45 pages.
- [6] Jared Culbertson and Kirk Sturtz. A categorical foundation for Bayesian probability. *Applied Categorical Structures*, pages 1–16, 2012.
- [7] Vincent Danos and Ilias Garnier. Dirichlet is natural. *Electronic Notes in Theoretical Computer Science*, 319:137 – 164, 2015. MFPS XXXI.
- [8] Thomas S Ferguson. A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, pages 209–230, 1973.
- [9] M. Giry. A categorical approach to probability theory. In B. Banaschewski, editor, *Categorical Aspects of Topology and Analysis*, number 915 in LNM, pages 68–85. Springer-Verlag, 1981.
- [10] Amol Kapila, Bela Frigyik, and Maya R. Gupta. Introduction to the Dirichlet distribution and related processes. Technical Report UWEETR-2010-0006, University of Washington., 2010.
- [11] Alexander S Kechris. *Classical descriptive set theory*, volume 156 of *Graduate Text in Mathematics*. Springer, 1995.
- [12] Klaus Keimel and Gordon Plotkin. Mixed powerdomains for probability and nondeterminism. *Logical Methods in Computer Science*, 2015.
- [13] Dexter Kozen. Kolmogorov extension, martingale convergence, and compositionality of processes. Technical report, Computing and Information Science, Cornell University, December 2015.
- [14] Dexter Kozen, Kim G Larsen, Radu Mardare, and Prakash Panangaden. Stone duality for markov processes. In *LICS 2013*, pages 321–330, 2013.
- [15] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1998.

## A Construction of the multiset functor B

**Proposition A.1** *For  $X$  Polish, let  $B(X) \triangleq \coprod_{n \in \mathbb{N}} X^n/S_n$ , where  $X^n/S_n$  is the quotient of  $X^n$  under the obvious action of  $S_n$  on tuples with the quotient topology, i.e. the final topology for the quotient map  $q : X^n \twoheadrightarrow X^n/S_n$ .  $B(X)$  is Polish.*

**Proof.** We first shown that if  $Q$  is dense in  $X$ , then  $Q^n/S_n$  is dense in  $X^n/S_n$ : let  $U$  be an open set of  $X^n/S_n$ , then  $q^{-1}(U)$  is open in  $X^n$  and intersects  $Q^n$ , i.e. there exists  $(r_1, \dots, r_n) \in Q^n$  with  $(r_1, \dots, r_n) \in q^{-1}(U)$ , but this means that  $q(r_1, \dots, r_n) \in U$  and  $q(r_1, \dots, r_n) \in Q^n/S_n$ . To see that it is completely metrisable, let  $d$  be a complete metric for  $X$ , and consider the metric on  $X^n/S_n$  given by:

$$d_q([x], [y]) = \min_{\pi \in S_n} d^n(x, \pi(y))$$

where  $[x], [y]$  represent the orbits of  $x, y \in X^n$  respectively, and  $d^n$  is the product metric given by

$$d^n((x_1, \dots, x_n), (y_1, \dots, y_n)) = \left( \sum_i d(x_i, y_i)^p \right)^{\frac{1}{p}} \quad (\text{A.1})$$

for some  $0 < p < \infty$  (any choice of  $p$  generates an equivalent topology on  $X^n$ ). Note that for any permutation  $\pi \in S_n$ ,  $d^n(x, y) = d^n(\pi(x), \pi(y))$  since this simply amounts to re-arranging the summands in Eq. (A.1). It is not immediately clear that  $d_q$  is well-defined or that it defines a metric. To see that it is well defined let  $x'$  be another representative of  $[x]$ , then by definition there exists  $\rho \in S_n$  such that  $\rho(x) = x'$ , and it follows that

$$\min_{\pi \in S_n} d^n(x', \pi(y)) = \min_{\pi \in S_n} d^n(\rho(x), \pi(y)) = \min_{\pi \in S_n} d^n(x, \rho^{-1}\pi(y)) = \min_{\pi \in S_n} d^n(x, \pi(y))$$

It follows that  $d_q$  is well-defined. Let us now check that it is a metric. For any  $x, y$  we clearly have  $d_q([x], [y]) \geq 0$  and  $d_q([x], [y]) = 0$  means that there exists  $\pi \in S_n$  such that  $d^n(x, \pi(y)) = 0$  i.e.  $x = \pi(y)$  since  $d^n$  is a metric, and it follows that  $[x] = [y]$ . For the symmetry, note that  $d^n$  is invariant under permutations of  $S_n$ , i.e.  $d^n(x, y) = d^n(\pi(x), \pi(y))$  since this simply rearranges the order of the summands in the product metric. It follows that

$$\begin{aligned} d_q([x], [y]) &= \min_{\pi \in S_n} d^n(x, \pi(y)) \\ &= \min_{\pi \in S_n} d^n(\pi^{-1}(x), y) && d^n \text{ invariant under } \pi^{-1} \\ &= \min_{\pi \in S_n} d^n(y, \pi^{-1}(x)) && d^n \text{ is symmetric} \\ &= d_q(y, x) \end{aligned}$$

Finally, we need to check the triangular inequality. Since  $d^n$  satisfies the triangular

inequality we have for any choice  $\pi_1, \pi_2 \in S_n$  that:

$$\begin{aligned} d^n(x, \pi_1(y)) &\leq d^n(x, \pi_1(z)) + d^n(\pi_2(z), \pi(x)) \\ &\leq d^n(x, \pi_2(z)) + d^n(z, \pi_2^{-1}\pi_1(x)) \quad d^n \text{ invariant under } \pi_2^{-1} \end{aligned}$$

and it follows that  $d_q([x], [y]) \leq d_q([x], [z]) + d_q([z], [y])$  since going through all the combinations  $\pi_2^{-1}\pi_1$  will exhaust the entire group  $S_n$ . The fact that  $(X^n/S_n, d_q)$  is complete follows from the fact that  $(X^n, d^n)$  is. Let us prove that  $d_q$  induces the topology of  $X^n/S_n$ . Let us take an open set  $U$  in  $X^n/S_n$ . By definition  $q^{-1}(U)$  is open in  $X^n$ , and can therefore be written as a union of open balls (for the metric  $d$ )  $U = \bigcup_i B_{d^n}(x_i, \epsilon_i)$ . By definition  $q^{-1}(U)$  is invariant under permutation, so

$$\begin{aligned} q^{-1}(U) &= \bigcup_{\pi \in S_n} \pi(q^{-1}(U)) = \bigcup_{\pi \in S_n} \pi \left( \bigcup_i B_{d^n}(x_i, \epsilon_i) \right) = \bigcup_{\pi \in S_n} \bigcup_i \pi(B_{d^n}(x_i, \epsilon_i)) \\ &= \bigcup_i \bigcup_{\pi \in S_n} \pi(B_{d^n}(x_i, \epsilon_i)) \end{aligned}$$

since direct images commute with unions. It follows from the fact that each  $\pi$  is an homeomorphism that  $\bigcup_{\pi \in S_n} \pi(B_{d^n}(x_i, \epsilon_i))$  is open in  $X^n$ . Moreover,  $\bigcup_{\pi \in S_n} \pi(B_{d^n}(x_i, \epsilon_i))$  is by construction invariant under permutation, so

$$q^{-1}(q(\bigcup_{\pi \in S_n} \pi(B_{d^n}(x_i, \epsilon_i)))) = \bigcup_{\pi \in S_n} \pi(B_{d^n}(x_i, \epsilon_i))$$

and therefore each  $q(\bigcup_{\pi \in S_n} \pi(B_{d^n}(x_i, \epsilon_i)))$  is an open in  $X^n/S_n$ . We conclude by observing that  $q(\bigcup_{\pi \in S_n} \pi(B_{d^n}(x_i, \epsilon_i))) = B_{d_q}(q(x_i), \epsilon_i)$  and that

$$q^{-1}(B_{d_q}(q(x_i), \epsilon_i)) = q^{-1}(q(\bigcup_{\pi \in S_n} \pi(B_{d^n}(x_i, \epsilon_i)))) = \bigcup_{\pi \in S_n} \pi(B_{d^n}(x_i, \epsilon_i))$$

is open in  $X^n$ . Therefore, the balls  $B_{d_q}(q(x_i), \epsilon_i)$  are open in  $X^n/S_n$ , and since direct images commute with unions it is not difficult to see that  $U = \bigcup_i B_{d_q}(q(x_i), \epsilon_i)$  is a union of opens from the basis generated by the metric. Since each  $X^n/S_n$  is Polish and since **Pol** has countable coproducts,  $B(X)$  is Polish.  $\square$

## B Properties of the functors B and V

**Proposition B.1** *The multiset functor B preserves injections, embeddings and intersections.*

**Proof.** Note first that  $B(i)$  is defined component-wise i.e. via  $B_n(i) : B^n/S_n \rightarrow X^n/S_n$  injecting an equivalence class of  $n$ -tuples of element of  $B$  in  $X^n/S_n$ . The fact that  $B(i)$  is injective follows from the fact that every component  $B_n(i)$  is. Similarly to show that  $B(i)$  is an embedding it is enough to show that each  $B_n(i)$  is. To see that this is the case we need to show that for every open  $U$  of  $B^n/S_n$  there exists an open  $V$  of  $X^n/S_n$  such that  $U = V \cap B^n/S_n$  and conversely that every subset of this shape is open in  $B^n/S_n$ . We write  $p_n : B^n \rightarrow B^n/S_n$  and  $q_n : X^n \rightarrow X^n/S_n$ .

For the first direction, let  $U$  be open in  $B^n/S_n$ , it follows that  $p_n^{-1}(U)$  is open in  $B$ , and thus that there exists an open  $V$  of  $X^n$  such that  $p_n^{-1}(U) = B^n \cap V$ . If we can choose  $V$  to be closed under permutation we are done. Every permutation is a bijective isometry and thus a homeomorphism, and thus an open map, i.e.  $\pi(V)$  is open for every  $\pi \in S_n$ . It follows that

$$V^* = \bigcup_{\pi \in S_n} \pi(V)$$

is open and closed under permutations (this procedure amounts to taking all the reflections of tuples along the diagonal). It follows that  $q_n^{-1}(q_n[V^*]) = V^*$  and  $q_n(V^*)$  is thus open in  $X^n/S_n$ . Moreover since  $B^n \cap V$  is already closed under permutations  $B^n \cap V = B^n \cap V^*$ , and therefore  $U = B^n/S_n \cap q_n(V^*)$ . For the opposite direction, let  $U$  be open in  $X^n/S_n$  and consider  $U \cap B^n/S_n$ , it is clear that

$$p_n^{-1}(U \cap B^n/S_n) = p_n^{-1}(U) \cap p_n^{-1}(B^n/S_n) = (q_n^{-1}(U) \cap B) \cap B = q_n^{-1}(U) \cap B$$

which is open in  $B^n$  since  $q_n(U)$  is open in  $X^n$ .

For intersections, we proceed as in Proposition 5.1. Let  $j_1, j_2 : A, B \rightarrow X$  be two embeddings, let  $p_1 : A \cap B \rightarrow A$  and  $p_2 : A \cap B \rightarrow B$  be the corresponding embeddings and consider  $\mu \in \mathbf{B}A, \nu \in \mathbf{B}B$  such that  $\mathbf{B}j_1(\mu) = \mathbf{B}j_2(\nu)$ . We define  $\lambda \in \mathbf{B}(A \cap B)$

$$\lambda(x) = \mu(p_1(x)) = \nu(p_2(x))$$

We check that the last equality holds in exactly the same way as in the proof of Proposition 5.1, and the rest of the proof also follows identically.  $\square$

**Proposition B.2** *The Vietoris functor  $\mathbf{V}$  preserves monomorphisms, embeddings and intersections.*

**Proof.** It is clear that  $\mathbf{V}$  preserves injective maps. To see that it preserves embeddings, consider an element of the basis of the topology on  $\mathbf{V}(X)$ , i.e. an element of the form (Kechris [11] I, 4.F)

$$W = \{K \in \mathbf{V}(X) \mid K \subseteq U_0 \& K \cap U_1 \neq \emptyset \& \dots \& K \cap U_n \neq \emptyset\}$$

for  $U_0, \dots, U_n$  opens in  $X$ . It follows that

$$\begin{aligned} W \cap \mathbf{V}(B) \\ = \{K \in \mathbf{V}(B) \mid K \subseteq (U_0 \cap B) \& K \cap (U_1 \cap B) \neq \emptyset \& \dots \& K \cap (U_n \cap B) \neq \emptyset\} \end{aligned}$$

which is an element of the basis of the topology of  $\mathbf{V}(B)$ , since elements of the shape  $U_i \cap B$  are precisely the opens of  $B$ . Conversely therefore, starting from an element  $W'$  of this shape it is clear that by removing all the intersections with  $B$  we get an element  $W$  of the basis of the topology on  $\mathbf{V}(X)$  such that  $W \cap \mathbf{V}(B) = W'$ , and  $\mathbf{V}$  thus preserves embeddings.

For intersections, let  $j_1, j_2 : A, B \rightarrow X$  be two embeddings, let  $p_1 : A \cap B \rightarrow A$  and  $p_2 : A \cap B \rightarrow B$  be the corresponding embeddings and consider  $K_A \in \mathbf{V}A, K_B \in \mathbf{V}B$  such that  $\mathbf{V}j_1(K_A) = \mathbf{V}j_2(K_B)$ , i.e. such that  $j_1[K_A] = j_2[K_B]$ . This means that



$K = K_A = K_B$  is a subset of  $A \cap B$ . To see that it is compact in  $A \cap B$ , let  $\bigcup_i U_i \supseteq K$  be an open cover: for each  $i$  either  $U_i$  is of the form  $p_1^{-1}(V_i)$  for some  $V_i$  open in  $A$ , or it is of the form  $p_2^{-1}(V_i)$  for some  $V_i$  open in  $B$ . In the latter case, since  $j_2$  is an embedding, there exists  $W_i$  open in  $C$  such that  $U_i = p_2^{-1}(j_2^{-1}(W_i))$ , but then  $U_i = p_1^{-1}(j_1^{-1}(W_i))$ , which means that we can assume without loss of generality that for each  $i$  the element  $U_i$  of the cover is of the form  $p_1^{-1}(V_i)$  for some  $V_i$  open in  $A$ . It is easy to see that  $V_i$  is an open cover of  $K$  in  $A$ , from which we can extract a finite sub-cover, whose inverse image under  $p_1$  will be a finite sub-cover of  $K$  in  $A \cap B$ . It follows that  $\forall A \cap \forall B \simeq \forall(A \cap B)$  as sets, and since  $\forall$  preserves embeddings, they are also homeomorphic.  $\square$

**Proposition B.3**  $\mathbf{B}$  is  $\mathbf{Pol}_f$ -continuous.

**Proof.** Let  $X_i, i \in I$  be a ccd of  $\mathbf{Pol}_f$  objects. We show  $\lim \mathbf{B}X_i = \mathbf{B}(\lim X_i)$ . For this we need to show that the unique continuous map  $u : \mathbf{B}(\lim X_i) \rightarrow \lim \mathbf{B}X_i$  is a homeomorphism. To show this will show that it is bijective and open. We start by defining an inverse  $\phi : \lim \mathbf{B}X_i \rightarrow \mathbf{B}(\lim X_i)$ . Since the set underlying the limits are computed in **Set**, showing that  $\phi$  exists and is an inverse as a function will be enough to prove that  $u$  is bijective. We can assume w.l.o.g. that the morphisms between the finite Polish spaces  $X_i$  are surjective.

Given a ‘thread’  $(\mu_i)_{i \in I} \in \lim \mathbf{B}X_i$  we need to define a finitely supported multiset on the threads  $(x_i)_{i \in I} \in \lim X_i$ . For the thread  $(\mu_i)$  consider the projective system of supports  $(\text{supp}(\mu_i))_{i \in I}$  together with the obvious restrictions  $f_{ij} \upharpoonright \text{supp}(\mu_i)$  of the connecting maps  $f_{ij} : X_i \rightarrow X_j$  which are also surjective. We claim that  $\lim \text{supp}(\mu_i)$  is finite and forms the support of the multiset  $\phi((\mu_i)_{i \in I})$  on  $\lim X_i$ . We make the following observation:

- (i) Each support is finite
- (ii) The size of the support cannot increase by following the connecting arrows, since they are surjective.
- (iii) The total mass  $k$  of  $\mu_i, i \in I$  is constant throughout the thread because  $\mathbf{B}$  applied to a connecting morphism preserves the total mass of a multiset.
- (iv) The cardinality of the set  $\text{supp}(\mu_i)$  is bounded by  $k$  since we cannot assign a weight less than one to any element in the support.
- (v) There exists an  $i \in I$  after which the cardinality of  $\text{supp}(\mu_i)$  remains constant, i.e. such that  $|\text{supp}(\mu_k)| = |\text{supp}(\mu_j)|$  for each  $j > k$ . If this wasn’t the case it would contradict the previous points.

Thus let  $k$  be such that  $|\text{supp}(\mu_k)| = |\text{supp}(\mu_j)|$  for each  $j > k$ , we claim that  $p_k : \lim \text{supp}(\mu_i) \rightarrow \text{supp}(\mu_k)$  is a bijection. It is surjective since the connecting morphisms in the diagram are surjective. If  $(x_i)_{i \in I}, (y_i)_{i \in I}$  are two threads such that  $p_k(x_i) = p_k(y_i)$  then  $x_k = y_k$ . Now take any  $k' \in I$ , by co-directedness there exists  $j > k, k'$  and by assumption on  $k$ ,  $\text{supp}(\mu_k)$  and  $\text{supp}(\mu_j)$  have the same cardinality, i.e. the connecting morphism  $p_{jk}$  is bijective. There therefore exists a unique  $x_j \in \text{supp}(\mu_j)$  such that  $p_{jk}(x_j) = x_k = y_k$ , and it follows that both thread must go through the same element at  $k'$  too, for any  $k'$ , which shows that  $p_k$  is

injective. We define  $\phi((\mu_i)_{i \in I})$  as the multiset on  $\lim X_i$  defined by:

$$(x_i)_{i \in I} \mapsto \begin{cases} 0 & \text{if } (x_i)_{i \in I} \notin \lim \text{supp}(\mu_i) \\ \mu_k(x_k) & \text{else (where } k \text{ is defined as above)} \end{cases}$$

We need to show that the definition is independent of the choice of  $k$ . Consider another index  $k'$  such that  $|\text{supp}(\mu_{k'})| = |\text{supp}(\mu_j)|$  for each  $j > k'$ . Again by co-directedness there exists  $j > k, k'$ . We now calculate:

$$\begin{aligned} \mu_k(x_k) &= \mu_k(f_{jk}(x_j)) = \mathbf{B}f_{jk}(\mu_j)(f_{jk}(x_j)) = \mu_j(x_j) = \mathbf{B}f_{jk'}(\mu_j)(f_{jk'}(x_j)) \\ &= \mu_{k'}(x_{k'}) \end{aligned}$$

Let us now show that  $\phi$  thus defined is a left and right inverse to  $u$ . Given a multiset  $\mu \in \mathbf{B}(\lim X_i)$  on threads of  $\lim X_i$ ,  $u(\mu)$  is the thread of multisets  $\nu_i$  on  $X_i$  defined by  $\nu_i(x) = \mu[p_i^{-1}(\{x\})]$ , i.e. the mass given by  $\mu$  to the set of threads going through  $x \in X_i$ . This family forms a thread since for every  $f_{ij} : X_i \rightarrow X_j$  and  $y \in X_j$ ,

$$\nu_j(y) = \mu[p_j^{-1}(\{y\})] = \mu[p_i^{-1}(f_{ij}^{-1}(y))] = \nu_i(f_{ij}^{-1}(y)) = \mathbf{B}f_{ij}(\nu_i)(y)$$

For  $\mu \in \mathbf{B}(\lim X_i)$ , let  $u(\mu) = (\nu_i)_{i \in I}$ . The support  $\text{supp}(\nu_i)$  is given by the set  $Y_i \subseteq X_i$  of points traversed by a thread in the support of  $\mu$ , and it is therefore not hard to see that  $\lim \text{supp}(\nu_i)$  with the multiplicities defined by  $\phi$  is precisely  $\mu$ , i.e.  $\phi \circ u = \text{id}_{\mathbf{B}(\lim X_i)}$ . Conversely  $u \circ \phi = \text{id}_{\lim X_i}$  by universality of  $\lim X_i$ .

Finally, we show that the unique  $u : \mathbf{B}(\lim X_i) \rightarrow \lim \mathbf{B}X_i$  is a homeomorphism. We already know that it is continuous and bijective, so it remains to be shown that it is open. For this we must look at the topologies on  $\mathbf{B}(\lim X_i)$  and  $\lim \mathbf{B}X_i$ . In the former  $U$  is an open exactly when  $q_n^{-1}(U)$  is open in  $(\lim X_i)^n$  for each  $n$  where  $q_n : (\lim X_i)^n \twoheadrightarrow (\lim X_i)^n / S_n$ . Any subset  $U$  of  $\mathbf{B}(\lim X_i)$  can be written as an union of sets  $U_n$  in  $(\lim X_i)^n / S_n$ , so it is sufficient to show that  $u$  maps opens of  $(\lim X_i)^n / S_n$  (corresponding to sets of multisets of total mass  $n$ ) to opens in  $\lim \mathbf{B}X_i$ . It is not hard to check that  $U$  is open in  $(\lim X_i)^n / S_n$  iff there exists  $V$  open in  $(\lim X_i)^n$  such that  $q_n[V^*] = U$  where  $V^* = \bigcup_{\pi \in S_n} \pi[V]$ . To check that  $u(U)$  is open it is therefore enough to check that  $u \circ q_n \circ \pi[V]$  is open for any open in  $(\lim X_i)^n$ , and since  $\pi$  is a homeomorphism this really means checking that  $u \circ q_n[V]$  is open. By the definition of the product topology and of the topology on  $\lim X_i$  it is enough to check that  $u \circ q_n[Y_k^j]$  is open for  $Y_k^j$  the set of  $n$ -tuples of threads of  $\lim X_i$  whose  $j^{\text{th}}$  component goes through  $Y_k \subseteq X_k$ . The morphism  $q_n$  collapses such an  $n$ -tuple to a multiset on threads and  $q_n[Y_k^j]$  is the set of multisets of total mass  $n$  which assigns mass at least one to threads going through  $Y_k$ .

To check that  $u$  sends these open sets to open sets we need to describe the topology on the codomain. Fortunately it is much simpler. Since each  $X_i$  is finite,  $X_i^n / S_n$  is finite, and must therefore have the discrete topology. Since  $\mathbf{B}(X_i) = \coprod_n X_i^n / S_n$  is given the final topology for all the injections its topology must also be discrete. The topology on  $\lim \mathbf{B}(X_i)$  is thus generated by the opens of the shape  $p_i^{-1}(U_i)$  where  $U_i$  is any subset of  $\mathbf{B}(X_i)$  and  $p_i$  is the canonical projection.

We can now check that  $u$  is open. Let us denote  $q_n[Y_k^j] = Y_k^n$  the set of multisets of total mass  $n$  which assigns mass at least one to threads going through  $Y_k$ . It

gets mapped to a set  $B(p_k(Y_n^k))$  of multisets on  $X_k$ , which in turns defines  $u(Y_k^n) = p_k^{-1}(B(p_k(Y_n^k)))$  which is indeed open.  $\square$

**Proposition B.4** *Let  $(X_i)_{i \in I}$  be a ccd of compact spaces. Then  $\mathbf{V}(\lim X_i) \cong \lim \mathbf{V}X_i$*

**Proof.** Let  $(X_i)_{i \in I}$  be a ccd of compact Polish space; we must show that  $\mathbf{V} \lim X_i = \lim \mathbf{V}X_i$ . Let us first show that there exists a bijection between these sets. We write  $p_i : \lim X_i \rightarrow X_i$  for the canonical projections. We know that there exists a unique continuous map  $u : \mathbf{V} \lim X_i \rightarrow \lim \mathbf{V}X_i$ ; it takes a compact  $K$  of  $\lim X_i$  and maps it to the thread  $(p_i[K])_{i \in I}$  of  $\lim \mathbf{V}X_i$  (since the continuous image of a compact is compact). Let  $K, K'$  be two compacts of  $\lim X_i$  such that  $p_i[K] = p_i[K']$  for every  $i \in I$ , for every thread  $(x_i)_{i \in I}$  in  $\lim X_i$  it is clear that  $(x_i) \in K$  iff  $p_i(x_i) \in p_i[K]$  iff  $p_i(x_i) \in p_i[K']$  iff  $(x_i) \in K'$  and thus  $u$  is injective.

We now define an inverse map  $\phi : \lim \mathbf{V}X_i \rightarrow \mathbf{V} \lim X_i$  as follows. For each thread of compacts  $(K_i)_{i \in I}$  in  $\lim \mathbf{V}X_i$ , since each  $X_i$  is Hausdorff, each  $K_i$  is closed and thus  $p_i^{-1}(K_i)$  is a closed subset of  $\lim X_i$ . We define

$$\phi((K_i)_{i \in I}) = \bigcap_i p_i^{-1}(K_i)$$

To see that this is well-defined, we need to show that  $\phi((K_i)_{i \in I})$  is compact. Since each  $p_i^{-1}(K_i)$  is closed, their intersection  $\phi((K_i)_{i \in I})$  is closed. We also know that since each  $X_i$  is compact  $\lim X_i$  is a closed subspace of the product  $\prod X_i$  which is compact by Tychonoff's theorem. It follows that  $\lim X_i$  is compact, and since  $\mathbf{V}$  sends compacts to compacts (Kechris Theorem 4.26),  $\mathbf{V} \lim X_i$  is compact. Finally since  $\phi((K_i)_{i \in I})$  is closed in a compact it is itself compact.

To see that  $\phi$  is a left inverse of  $u$ , start with  $K \in \mathbf{V} \lim X_i$ ,  $u(K) = (p_i[K])_{i \in I}$  and

$$\phi(u(K)) = \bigcap_i p_i^{-1}(p_i[K])$$

Let  $(x_i)$  be a thread in  $K$ , then clearly  $p_i((x_i)) = x_i \in p_i[K]$  for all  $i$ , and thus  $(x_i) \in \phi(u(K))$ . Conversely, let  $(x_i)$  be a thread in  $\phi(u(K))$  then by definition of  $\phi$ ,  $p_i((x_i)) \in p_i[K]$  for every  $i$ , i.e.  $x_i \in p_i[K]$  for every  $i$ , i.e.  $(x_i) \in K$ , and it follows that  $\phi \circ u = id_{\mathbf{V} \lim X_i}$ . Conversely,  $\phi$  is a right inverse since  $u \circ \phi = id_{\lim \mathbf{V}X_i}$  by universality of  $u$ . We have thus established that  $u$  is bijective.

Finally, since  $u : \mathbf{V} \lim X_i \rightarrow \lim \mathbf{V}X_i$  is a continuous bijection with a compact domain and a Hausdorff codomain, it is a homeomorphism, which concludes the proof.  $\square$

# Complete Elgot Monads and Coalgebraic Resumptions<sup>†</sup>

Sergey Goncharov<sup>1,\*</sup> Stefan Milius<sup>2</sup> Christoph Rauch<sup>3,\*</sup>

*Lehrstuhl für Theoretische Informatik, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany*

---

## Abstract

*Monads* are used to abstractly model a wide range of computational effects such as nondeterminism, statefulness, and exceptions. *Complete Elgot monads* are monads that are equipped with a (uniform) iteration operator satisfying a set of natural axioms, which allows to model *iterative computations* just as abstractly. It has been shown recently that extending complete Elgot monads with free effects (e.g. operations of sending/receiving messages over channels) canonically leads to *generalized coalgebraic resumption monads*, which were previously used as semantic domains for non-wellfounded guarded processes. In this paper, we continue the study of complete Elgot monads and their relationship with generalized coalgebraic resumption monads. We give a characterization of the Eilenberg-Moore algebras of the latter. In fact, we work more generally with Uustalu's *parametrized monads*; we introduce complete Elgot algebras for a parametrized monad and we prove that they form an Eilenberg-Moore category. This is further used for establishing a characterization of complete Elgot monads as those monads whose algebras are coherently equipped with the structure of complete Elgot algebras for the parametrized monads obtained from generalized coalgebraic resumption monads.

*Keywords:* Complete Elgot monad, complete Elgot algebra, resumption monad, uniform iteration

---

## 1 Introduction

One traditional use of monads in computer science, stemming from the seminal thesis of Lawvere [20], is as a tool for algebraic semantics where monads arise as a high-level metaphor for (clones of) equational theories. More recently, Moggi proposed to associate monads with *computational effects* and use them as a generic tool for denotational semantics [22], which later had a considerable impact on the design of functional programming languages, most prominently Haskell [1]. Finally, in the first decade of the new millennium, Plotkin and Power reestablished the connection between computational monads and algebraic theories in their theory of *algebraic effects* [23, 24].

---

<sup>†</sup> Full version is available at <http://arxiv.org/abs/1603.02148>

\* Supported by Deutsche Forschungsgemeinschaft (DFG) under project SCHR 1118/8-1

<sup>1</sup> Email: [Sergey.Goncharov@fau.de](mailto:Sergey.Goncharov@fau.de)

<sup>2</sup> Email: [mail@stefan-milius.eu](mailto:mail@stefan-milius.eu) Supported by Deutsche Forschungsgemeinschaft (DFG) under project MI 717/5-1

<sup>3</sup> Email: [Christoph.Rauch@fau.de](mailto:Christoph.Rauch@fau.de)

We use the outlined view to study the notion of *iteration* as a concept that has a well-established algebraic meaning and which is very relevant in the context of computational effects. On the technical level our present work can be viewed as a continuation of the previous extensive work on monads with iteration [2, 5, 7] having its roots in the work of Elgot [12] and Bloom and Ésik [10] on iteration theories.

More specifically, we are concerned with a particular construction on monads: given a monad  $\mathbb{T}$  and a functor  $\Sigma$ , we assume the existence of the coalgebra

$$T_{\Sigma}X = \nu\gamma. T(X + \Sigma\gamma) \quad (\star)$$

for each object  $X$  (these final coalgebras exist under mild assumptions on  $T$ ,  $\Sigma$ , and the base category). It is known [28] that  $T_{\Sigma}$  extends to a monad  $\mathbb{T}_{\Sigma}$  and we call the latter the *generalized coalgebraic resumption monad*.

Intuitively,  $(\star)$  is a generic semantic domain for systems combining *extensional* (via  $\mathbb{T}$ ) and *intensional* (via  $\Sigma$ ) features with iteration. To make this intuition more precise, consider the following simplistic

**Example 1.1** Let  $A = \{a, b\}$  be an alphabet of *actions*. Then the following system of equations specifies *processes*  $x_1, x_2, x_3$  of *basic process algebra (BPA)*:

$$x_1 = a \cdot (x_2 + x_3) \quad x_2 = a \cdot x_1 + b \cdot x_3 \quad x_3 = a \cdot x_1 + \checkmark$$

We can think of this specification as a map  $P \rightarrow T(\{\checkmark\} + \Sigma P)$  where  $P = \{x_1, x_2, x_3\}$ ,  $\Sigma = A \times -$  and  $T = \mathcal{P}_{\omega}$  is the finite powerset monad. Using the standard approach [26] we can *solve* this specification by finding a map  $P \rightarrow T_{\Sigma}\{\checkmark\}$  that assigns to every  $x_i$  the corresponding semantics over the domain of possibly non-wellfounded trees  $T_{\Sigma}\{\checkmark\} = \nu\gamma. \mathcal{P}_{\omega}(\{\checkmark\} + A \times \gamma)$ . The crucial fact here is that the original system is *guarded*, i.e. every recursive call of a variable  $x_i$  is preceded by an action. This implies that the given recursive system has a unique solution.

If the guardedness assumption is dropped, solutions may fail to be unique, but it is possible to introduce a notion of *canonical solution* if the Kleisli category of the monad  $\mathbb{T}$  is enriched in the category of complete partial orders, or more generally, if  $\mathbb{T}$  is a *complete Elgot monad*. A monad  $\mathbb{T}$  is a complete Elgot monad if it is equipped with an iteration operator that assigns to every morphism of the form  $f : X \rightarrow T(Y + X)$  a *solution*  $f^{\dagger} : X \rightarrow TY$  satisfying a certain well-established set of equational axioms of iteration and also *uniformity* [27] (e.g.  $\mathcal{P}_{\omega}$  is not a complete Elgot monad, but the countable powerset monad  $\mathcal{P}_{\omega_1}$  is). The central result of the recent work [14] is that whenever  $\mathbb{T}$  is a complete Elgot monad then so is the transformed monad  $(\star)$ . In particular, this allows for canonical solutions of recursive equations over processes (in the sense of Example 1.1) whenever recursive equations over  $\mathbb{T}$  are solvable.

In the present paper we investigate the relationship between guarded and unguarded iteration, which are implemented via generalized coalgebraic resumption monads and complete Elgot monads, respectively. As an auxiliary abstraction device, we use the notion of a *parametrized monad* introduced by Uustalu [28], i.e. a bifunctor  $\# : \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$  such that for every object  $X$ , the functor  $- \# X$  is a monad. For example, the bifunctor  $X \# Y = T(X + \Sigma Y)$  in  $(\star)$  yields a parametrized monad.

Following [5], we introduce *complete Elgot algebras* for a parametrized monad  $\#$ , which are algebras for  $\#$  equipped with an iteration operator satisfying simplified versions of the axioms of complete Elgot monads. However, in contrast to the latter complete Elgot  $\#$ -algebras omit any form of the Bekić law that states how mutually recursive definitions are solved. We then prove that for every object  $X$  the final coalgebra  $\nu\gamma.X \# \gamma$  is equivalently a free complete Elgot  $\#$ -algebra on  $X$  and that the category of Eilenberg-Moore algebras for the ensuing monad  $\nu\gamma.- \# \gamma$  is isomorphic to the category of complete Elgot  $\#$ -algebras.

Furthermore, we show that for every complete Elgot monad  $\mathbb{T}$ , every free  $\mathbb{T}$ -algebra  $TZ$  canonically extends to a complete Elgot  $\#$ -algebra for  $X \# Y = T(X + Y)$ . This situation can be roughly summarized as follows:

$$(\nu\gamma.- \# \gamma)\text{-algebras} \quad \cong \quad \text{complete Elgot } \# \text{-algebras} \quad \supseteq \quad \text{free } \mathbb{T}\text{-algebras}$$

From the perspective of Example 1.1 this connection can be regarded as follows. Since  $\Sigma = \text{Id}$ , the set of guards consists of only one action, which can be understood as *delaying*. Now, the inclusion of  $\mathbb{T}$ -algebras into  $(\nu\gamma.- \# \gamma)$ -algebras essentially means that complete Elgot monads interpret staged, possibly infinite, guarded processes over  $\mathbb{T}$  by forgetting the guards.

Suppose that, conversely, we have any monad  $\mathbb{T}$  such that the above inclusion holds in the sense that  $\mathbb{T}$ -algebras are coherently equipped with structures of complete Elgot  $\#$ -algebras. Then  $\mathbb{T}$  is equipped with an iteration operator satisfying a set of axioms that are weaker than the axioms of complete Elgot monads; the ensuing notion is that of a *weak complete Elgot monad*.

The paper is organized as follows. After categorical preliminaries (Section 2) we present and discuss complete Elgot monads in Section 3. In Section 4 we introduce algebras and complete Elgot algebras for a parametrized monad  $\#$ ; next, in Section 5, we show that the category of complete Elgot  $\#$ -algebras is isomorphic to the Eilenberg-Moore category of the monad  $\nu\gamma.- \# \gamma$  (Theorem 5.7); furthermore, we show that a free complete Elgot  $\#$ -algebra on  $X$  is equivalent to the final coalgebra  $\nu\gamma.X \# \gamma$  (Theorem 5.9). Finally, in Section 6 we apply the developed results to characterize complete Elgot monads as those whose algebras are coherently equipped with complete Elgot algebra structures (Theorem 6.4 and 6.6).

**Further Related Work.** Algebras for parametrized monads were introduced in [4, 6] albeit for the special case of a *base*, i.e. a finitary parametrized monad on a locally finitely presentable category. Loc. cit. also introduces iterative base algebras which are algebras for a base having unique solutions of finitary recursive equations. Complete Elgot algebras for an endofunctor  $H$  were introduced in [5], and it was proved that they form the Eilenberg-Moore category of the monad  $T$  obtained by taking the final coalgebras  $TX = \nu\gamma.(X + H\gamma)$ ; this is the free completely iterative monad on  $H$  (see [2]). Since  $X \# Y = X + HY$  is a parametrized monad, our notion of complete Elgot algebras generalizes the previous notion to the level of parametrized monads and it extends iterative algebras by considering an iteration operation subject to certain axioms in lieu of unique solutions. Our Theorem 5.7 generalizes [5, Theorem 5.8].

The study of monads with an iteration operator is inspired by Bloom and Ésik's iteration theories [10]. Extending this from Lawvere theories (i.e. finitary monads

on **Set**) to monads on more general categories has led to the notion of Elgot monad introduced in [7]. While iteration theories and Elgot monads study an iteration operator for recursive equations with finitely many recursion variables, complete Elgot monads [14] are equipped with an iteration operator for all (finitary and infinitary) recursive equations.

## 2 Preliminaries

We assume that readers are familiar with basic category theory [21]; we write  $|\mathbf{C}|$  for the class of objects of a category  $\mathbf{C}$  and  $f : X \rightarrow Y$  for morphisms in  $\mathbf{C}$ . We often omit indexes, e.g. on natural transformations, if they are clear from the context.

In this paper we work with an ambient category  $\mathbf{C}$  with finite coproducts. We denote by  $\text{inl}$  and  $\text{inr}$  the left- and right-hand coproduct injections from  $X$  and  $Y$  to  $X + Y$ , and  $[f, g] : X + Y \rightarrow Z$  the is the *copair* of  $f : X \rightarrow Z$  and  $g : Y \rightarrow Z$ , i.e. the unique morphism with  $[f, g] \text{inl} = f$  and  $[f, g] \text{inr} = g$ . The codiagonal is denoted by  $\nabla = [\text{id}, \text{id}] : X + X \rightarrow X$  as usual.

We consider *monads* on  $\mathbf{C}$  given in the form of *Kleisli triples*  $\mathbb{T} = (T, \eta, -^*)$  where  $T$  is an endomap on  $|\mathbf{C}|$ ,  $\eta$ , called *monad unit*, is a family of morphisms  $\eta_X : X \rightarrow TX$  indexed over  $|\mathbf{C}|$ , and (*Kleisli*) *lifting* assigns to each  $f : X \rightarrow TY$  a morphism  $f^* : TX \rightarrow TY$  such that the following laws hold:

$$\eta^* = \text{id}, \quad f^* \eta = f, \quad (f^* g)^* = f^* g^*.$$

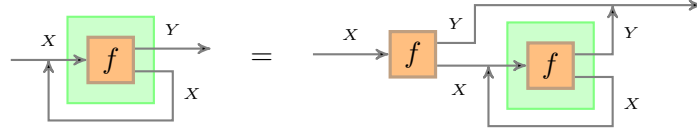
This is equivalent to the definition of a monad as a triple  $(T, \eta, \mu)$  that consists of a functor  $T$  and two natural transformations, the monad unit  $\eta : \text{Id} \rightarrow T$  and the *monad multiplication*  $\mu : TT \rightarrow T$  [21]. In particular, given a Kleisli triple,  $\mu = \text{id}^*$  yields the monad multiplication,  $\eta$  extends to a natural transformation, and  $T$  to an endofunctor with morphism mapping  $Tf = (\eta f)^*$ . The *Kleisli category*  $\mathbf{C}_{\mathbb{T}}$  of  $\mathbb{T}$  is formed by *Kleisli morphisms*  $X \rightarrow TY$ , i.e.  $\mathbf{C}_{\mathbb{T}}(X, Y) = \mathbf{C}(X, TY)$  with  $\eta_X$  as identity morphism on  $X$  and *Kleisli composition*: given Kleisli morphisms  $f : X \rightarrow TY$  and  $g : Y \rightarrow TZ$  we have

$$f \diamond g = \left( X \xrightarrow{f} TY \xrightarrow{g^*} TZ \right).$$

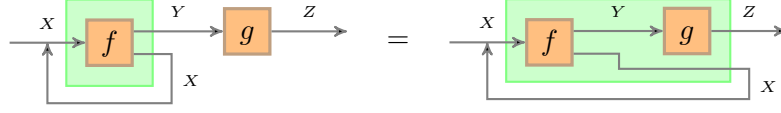
We write  $f : X \multimap Y$  for a Kleisli morphism  $f : X \rightarrow TY$ .

The forgetful functor from  $\mathbf{C}_{\mathbb{T}}$  to  $\mathbf{C}$  has a left adjoint sending any  $f : X \rightarrow Y$  to  $\underline{f} = \eta f : X \rightarrow TY$ . Like any left adjoint, this functor preserves colimits, and in particular coproducts. Since  $|\mathbf{C}| = |\mathbf{C}_{\mathbb{T}}|$ , this implies that coproducts in  $\mathbf{C}_{\mathbb{T}}$  exist and are lifted from  $\mathbf{C}$ . Explicitly,  $\underline{\text{inl}} = \eta \text{inl} : X \multimap X + Y$ ,  $\underline{\text{inr}} = \eta \text{inr} : Y \multimap X + Y$  are the coproduct injections in  $\mathbf{C}_{\mathbb{T}}$  and  $[f, g] : A + B \multimap C$  (formed in  $\mathbf{C}$ ) is the copair of  $f : A \multimap C$  and  $g : B \multimap C$  in  $\mathbf{C}_{\mathbb{T}}$ . We denote by  $f \oplus g : A + B \multimap A' + B'$  the coproduct of morphisms  $f : A \multimap A'$  and  $g : B \multimap B'$  in  $\mathbf{C}_{\mathbb{T}}$ . Besides  $\mathbf{C}_{\mathbb{T}}$ , we consider the category  $\mathbf{C}^{\mathbb{T}}$  of (*Eilenberg-Moore*) *algebras* for  $\mathbb{T}$ , whose objects are pairs  $(A, a : TA \rightarrow A)$ , satisfying two laws:  $a \eta = \text{id}$  and  $a(Ta) = a \mu$ ; a  $\mathbb{T}$ -algebra morphism  $f$  from  $(A, a)$  to  $(B, b)$  is a morphism  $f : A \rightarrow B$  such that  $f a = b T f$ . See [21] for more details.

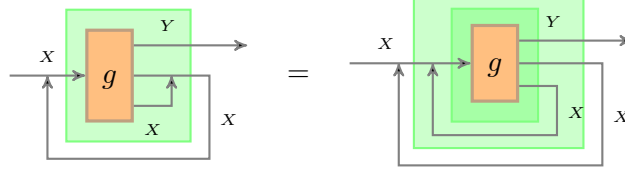
Fixpoint:



Naturality:



Codiagonal:



Uniformity:

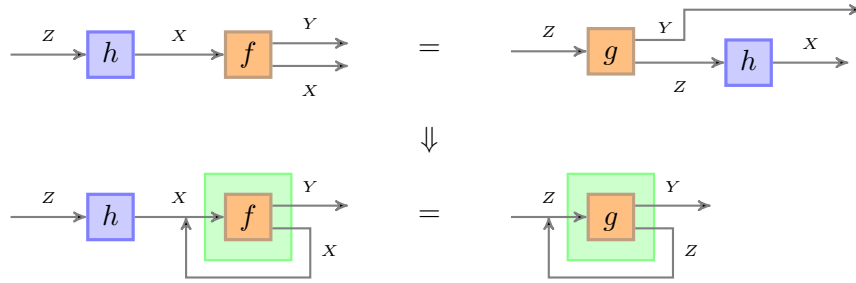


Fig. 1. Axioms of complete Elgot monads.

We will make use of standard facts on *coalgebras* for an endofunctor [25]. Given an endofunctor  $F : \mathbf{C} \rightarrow \mathbf{C}$ , an  $F$ -coalgebra is a pair  $(X, c)$  where  $X$  is an object of  $\mathbf{C}$  called the *carrier* of the coalgebra and  $c : X \rightarrow FX$  is a morphism called the *(transition) structure*. A coalgebra morphism  $f$  from  $(X, c)$  to  $(Y, d)$  is a morphism  $f : X \rightarrow Y$  such that  $df = (Ff)c$ . Coalgebras and their morphisms form a category. The final  $F$ -coalgebra, if it exists, is the terminal object in that category and is denoted by

$$\nu F \xrightarrow{\text{out}} F(\nu F).$$

By Lambek's lemma,  $\text{out}$  is an isomorphism, whose inverse  $\text{out}^{-1} : F(\nu F) \rightarrow \nu F$  can be obtained as  $\text{coit}(F \text{ out})$ , where for any coalgebra  $(X, f : X \rightarrow FX)$  we denote by  $\text{coit } f$  the unique coalgebra morphism  $X \rightarrow \nu F$  from  $X$  to the final coalgebra  $\nu F$ .

### 3 Complete Elgot Monads for Iteration

Complete Elgot monads are a slight generalization of Elgot monads from [7,8], which in turn, for the base category being **Set**, correspond precisely to those iteration theories of Bloom and Ésik [10] that satisfy the functorial dagger implication for base morphisms. In the following definition cited from [14] (for simplicity, we do not consider *strong* monads here because the possible presence of a strength has no bearing on our results), we follow the terminology of [9,27] where the same axioms



were considered in the dual setting of generic parametrized recursion.

**Definition 3.1 (Complete Elgot monads)** A complete Elgot monad is a monad  $\mathbb{T}$  equipped with an operator  $-^\dagger$ , called iteration, that assigns to each morphism  $f : X \multimap Y + X$  a morphism  $f^\dagger : X \multimap Y$  such that the following axioms hold:

- fixpoint:  $f^\dagger = [\eta, f^\dagger] \diamond f$ , for any  $f : X \multimap Y + X$ ;
- naturality:  $g \diamond f^\dagger = ((g \oplus \eta) \diamond f)^\dagger$  for any  $f : X \multimap Y + X$  and  $g : Y \multimap Z$ ;
- codiagonal<sup>4</sup>:  $([\eta, \text{inr}] \diamond g)^\dagger = g^{\dagger\dagger}$  for any  $g : X \multimap (Y + X) + X$ ;
- uniformity:  $f \diamond \underline{h} = (\eta \oplus \underline{h}) \diamond g$  implies  $f^\dagger \diamond \underline{h} = g^\dagger$  for any  $f : X \multimap Y + X$ ,  $g : Z \multimap Y + Z$  and  $h : Z \rightarrow X$ .

The above axioms of iteration can be comprehensibly represented in a flowchart-style as in Fig. 1. Here the feedback loops correspond to iteration and the coloured frames indicate the scope of the constructs being iterated. We believe that this presentation is rather well in touch with the intuition. For example, the naturality axiom expresses the fact that the scope of the iteration can be stretched to embrace a function post-processing the output of the terminating branch.

There is an obvious similarity between the axioms in Fig. 1 and the axioms of *traced monoidal categories* [18]. In fact, Hasegawa [16] proved that there is an equivalent presentation of a dagger operation satisfying the above axioms in terms of a uniform trace operator w.r.t. coproducts (actually, Hasegawa worked in the dual setting with products). Note that the present axioms make use of coproduct injections and the codiagonal morphism, while the trace axioms can be formulated more generally for any monoidal product.

One standard source of examples for complete Elgot monads is a suitable enrichment of the Kleisli category  $\mathbf{C}_{\mathbb{T}}$  over complete partial orders.

**Example 3.2 ( $\omega$ -continuous monads)** An  $\omega$ -continuous monad is a monad  $\mathbb{T}$  such that the Kleisli category  $\mathbf{C}_{\mathbb{T}}$  is enriched over the category  $\mathbf{Cppo}$  of  $\omega$ -complete partial orders with bottom  $\perp$  and (nonstrict) continuous maps; moreover, composition in  $\mathbf{C}$  is required to be left strict and composition in  $\mathbf{C}_{\mathbb{T}}$  right strict:  $\perp f = \perp$ ,  $f \diamond \perp = \perp$ ; equivalently,  $\perp$  is a *constant* of  $\mathbb{T}$ . We also assume that coproducts in  $\mathbf{C}_{\mathbb{T}}$  are  $\mathbf{Cppo}$ -enriched; for this it suffices that copairing is monotone in both arguments. It then follows that it is also continuous; for  $\bigsqcup_i [f_i, g]$  is a morphism satisfying  $(\bigsqcup_i [f_i, g]) \text{inl} = \bigsqcup_i f_i$  and  $(\bigsqcup_i [f_i, g]) \text{inr} = g$  by continuity of composition, whence  $\bigsqcup_i [f_i, g] = [\bigsqcup_i f_i, g]$ . Similarly, one shows continuity in the second argument. (Note that monotonicity is used only so that the  $[f_i, g]$  form an  $\omega$ -chain provided that the  $f_i$  do.)

It is shown in [14] that an  $\omega$ -continuous monad is a complete Elgot monad with  $e^\dagger$  calculated as the least fixed point of the map  $f \mapsto [\eta, f] \diamond e$ . This yields the powerset monad  $\mathcal{P}$ , the *Maybe-monad*  $(- + 1)$ , or the nondeterministic state monad  $\mathcal{P}(- \times S)^S$  as examples of complete Elgot monads on  $\mathbf{Set}$ . The *lifting monad*  $(-)_\perp$  is a complete Elgot monad on the category of complete partial orders without bottom.

<sup>4</sup> The codiagonal axiom is often written as  $((\eta \oplus \nabla) \diamond g)^\dagger = g^{\dagger\dagger}$  implicitly alluding to the canonical isomorphism  $Y + (X + X) \cong (Y + X) + X$ .

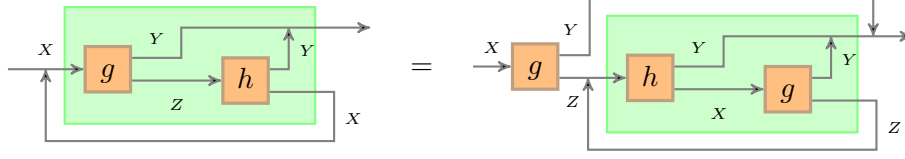


Fig. 2. Dinaturality axiom.

Another principal source of examples are *free complete Elgot monads* for which the iteration of guarded morphisms is uniquely defined.

**Example 3.3 (Free complete Elgot monads)** Suppose  $\mathbb{T}$  is the initial complete Elgot monad. It is shown in [14] that whenever the functor  $\mathbb{T}_\Sigma$  defined by (★) exists, it yields the *free complete Elgot monad on  $\Sigma$*  (note that the original  $\mathbb{T}$  is the free complete Elgot monad on the constant functor on the initial object of  $\mathbf{C}$ ). In **Set** (more generally, in any *hyperextensive category* [3]) the initial complete Elgot monad  $\mathbb{T}$  is the Maybe-monad  $- + 1$ .

**Example 3.4 (Capretta’s partiality monad)** One instructive example, not covered by the above cases is the coalgebraic resumption monad  $\nu\gamma. - + \gamma$ , studied by Capretta for modeling partiality in the intensional type theory [11]. Note that this example is not covered by Example 3.3, for that only states that  $\nu\gamma. T(X + \gamma)$  is a complete Elgot monad, provided  $\mathbb{T}$  is one (e.g. it follows that  $\nu\gamma. X + 1 + \gamma$  is a complete Elgot monad on **Set**), but  $T = \text{Id}$  is not a complete Elgot monad in any of the relevant examples.

We conjecture that  $T = \nu\gamma. - + \gamma$  can be shown to be a complete Elgot monad over any sufficiently rich (type-theoretic) universe; in particular, this can be easily seen in **Set**: Here  $TX$  explicitly evaluates to  $X \times \mathbb{N} + \{\perp\}$ , hence every  $TX$  can be ordered as a flat domain, i.e.  $x \sqsubseteq y$  iff  $x = \perp$  or  $x = y$ ; this easily extends to a **Cppo**-enrichment as required in Example 3.2 and hence gives rise to a complete Elgot monad structure on  $\mathbb{T}$ . Intuitively, every function  $f : X \rightarrow T(Y + X)$  either diverges or delivers a result together with the number of steps needed to compute it. The iteration  $f^\dagger$  sums the numbers occurring across the loop and in case of convergence delivers the sum together with the result value. Note that the number of unfoldings of  $f^\dagger$  in this process does not contribute to the result, which explains why the fixpoint identity indeed holds for  $\mathbb{T}$ .

In comparison to the previous work [14], Definition 3.1 remarkably drops the axiom of *dinaturality* (see Fig. 2). The reason is that this axiom turns out to be derivable, which is a fact that was recently discovered and formalized on the level of iteration theories [13]. Corollary 6 from *op. cit.* can be couched in present terms as follows (modulo the terminological change: *parameter identity* instead of *naturality*, *double dagger* instead of *codiagonal* and *dagger implication for base morphisms* instead of *uniformity*):

**Proposition 3.5 (Dinaturality)** *Given  $g : X \multimap Y + Z$  and  $h : Z \multimap Y + X$ , then*

$$([\text{inl}, h] \diamond g)^\dagger = [\eta, ([\text{inl}, g] \diamond h)^\dagger] \diamond g.$$

The codiagonal axiom in Definition 3.1 can equivalently be replaced by a form of

the well-known *Bekić identity*, see [10].

**Proposition 3.6 (Bekić identity)** *A complete Elgot monad  $\mathbb{T}$  is, equivalently, a monad satisfying the fixpoint, naturality and uniformity axioms (as in Definition 3.1), and the Bekić identity*

$$(T\alpha[f, g])^\dagger = [\eta, h^\dagger] \diamond [\text{inr}, g^\dagger],$$

where  $g : X \multimap (Z + Y) + X$ ,  $f : Y \multimap (Z + Y) + X$ ,  $h = [\eta, g^\dagger] \diamond f : Y \multimap Z + Y$ , with  $\alpha : (A + B) + C \rightarrow A + (B + C)$  being the obvious associativity isomorphism.

## 4 Parametrized Monads for Complete Elgot Algebras

In order to study complete Elgot monads and their algebras it is helpful to make a further abstraction step and generalize from monads to *parametrized monads* [28] (finitary parametrized monads are also called *bases* [4]), which are of independent interest.

**Definition 4.1 (Parametrized monad)** A *parametrized monad* over  $\mathbf{C}$  is a functor from  $\mathbf{C}$  to the category of monads over  $\mathbf{C}$  and monad morphisms. More explicitly, a parametrized monad is a bifunctor  $\# : \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$  such that for any  $X \in |\mathbf{C}|$ ,  $- \# X : \mathbf{C} \rightarrow \mathbf{C}$  is a monad, and for any  $f : X \rightarrow Y$ ,  $\text{id} \# f : Z \# X \rightarrow Z \# Y$  is the  $Z$ -component of a monad morphism from  $- \# X$  to  $- \# Y$ .

**Remark 4.2** The order of arguments in  $X \# Y$  is in agreement with [28] and differs from [4] where the notation  $Y \square X$  equivalent to the present  $X \# Y$  is used. We chose the order of arguments to ensure agreement with the type profile of the iteration operator  $-^\dagger$ , which is in turn in agreement with the expression  $(\star)$ .

Following [4] we will from now on denote the unit and multiplication of the monads  $- \# X$  by  $u_A^X : A \rightarrow A \# X$  and  $m_A^X : (A \# X) \# X \rightarrow A \# X$ , respectively.

**Example 4.3 (Parametrized monads)** We recall some standard examples of parametrized monads from [28]; further examples can be found e.g. in [6].

- (i) Whenever  $\mathbb{T} = (T, \eta, -^\star)$  is a monad and  $\Sigma$  is a functor,  $A \# X = T(A + \Sigma X)$  is a parametrized monad with the unit given by

$$u_A^X = \left( A \xrightarrow{\text{inl}} A + \Sigma X \xrightarrow{\eta_{A+\Sigma X}} T(A + \Sigma X) \right)$$

and the multiplication by

$$m_A^X = \left( T(T(A + \Sigma X) + \Sigma X) \xrightarrow{[\text{id}, \eta_{A+\Sigma X} \text{inr}]^\star} T(A + \Sigma X) \right).$$

Specifically, if  $\Sigma$  is the constant functor on an object  $E$  then  $X \# Y$  is the exception monad transformer with exceptions from  $E$  [22]. Another interesting special case is when  $\mathbb{T}$  is the identity monad (cf. Remark 4.8).

- (ii)  $A \# X = A \times X^\star$  is a parametrized monad with the unit and multiplication given by

$$u_A^X : a \mapsto (a, \varepsilon) \quad \text{and} \quad m_A^X : (a, w, v) \mapsto (a, wv),$$

where  $\varepsilon$  denotes the empty word and  $wv$  concatenation of words.

- (iii) Given a contravariant endofunctor  $H$ ,  $A \# X = A^{HX}$  is a parametrized monad with the unit and multiplication given by

$$u_A^X : a \mapsto \lambda x. a \quad \text{and} \quad m_A^X : (f : HX \rightarrow (HX \rightarrow A)) \mapsto \lambda x. f(x)(x).$$

This is a generalization of the well known *reader monad*, which can be recovered by instantiating  $H$  with a constant functor.

The following is a straightforward extension of the notion of an algebra for a base studied in [4] to arbitrary parametrized monads.

**Definition 4.4 (#-algebras)** Given a parametrized monad  $\# : \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$ , a  $\#$ -algebra is a pair  $(A, a)$  consisting of an object  $A$  of  $\mathbf{C}$ , and an algebra for the monad  $\#A$ , i.e. a morphism  $a : A \# A \rightarrow A$  satisfying

$$\begin{array}{ccc} A & \xrightarrow{u_A^A} & A \# A \\ \text{id} \searrow & & \downarrow a \\ & & A \end{array} \quad \begin{array}{ccc} (A \# A) \# A & \xrightarrow{a \# \text{id}} & A \# A \\ m_A^A \downarrow & & \downarrow a \\ A \# A & \xrightarrow{a} & A \end{array}$$

A morphism between  $\#$ -algebras  $(A, a)$  and  $(B, b)$  is a  $\mathbf{C}$ -morphism  $f : A \rightarrow B$  such that

$$\begin{array}{ccc} A \# A & \xrightarrow{a} & A \\ f \# f \downarrow & & \downarrow f \\ B \# B & \xrightarrow{b} & B \end{array}$$

For our leading example  $X \# Y = T(X + \Sigma Y)$  the category of  $\#$ -algebras can be described explicitly. Recall that a  $\mathbb{T}$ - $\Sigma$ -bialgebra in the sense of Kelly [19] is a triple  $(A, a, f)$  where  $a : TA \rightarrow A$  is a  $\mathbb{T}$ -algebra and  $f : \Sigma A \rightarrow A$  is a  $\Sigma$ -algebra.

**Proposition 4.5** *Let  $X \# Y = T(X + \Sigma Y)$  for a monad  $\mathbb{T}$  and a functor  $\Sigma$  on  $\mathbf{C}$ . Then  $\#$ -algebras are precisely  $\mathbb{T}$ - $\Sigma$ -bialgebras.*

**Proof (Sketch).** Given a  $\#$ -algebra  $\alpha : T(A + \Sigma A) \rightarrow A$  one forms two algebra structures

$$a = \left( TA \xrightarrow{T \text{inl}} T(A + \Sigma A) \xrightarrow{\alpha} A \right)$$

and  $b = \left( \Sigma A \xrightarrow{\text{inr}} A + \Sigma A \xrightarrow{\eta} T(A + \Sigma A) \xrightarrow{\alpha} A \right).$

A straightforward calculation then shows that  $a$  is a  $\mathbb{T}$ -algebra structure, whence  $(A, a, b)$  is a  $\mathbb{T}$ - $\Sigma$ -bialgebra.

Conversely, given any  $\mathbb{T}$ - $\Sigma$ -bialgebra  $a : TA \rightarrow A \leftarrow \Sigma A : b$  one forms

$$\alpha = \left( T(A + \Sigma A) \xrightarrow{T[\text{id}, b]} TA \xrightarrow{a} A \right).$$

Another straightforward computation establishes that this is the structure of a  $\#$ -algebra.

Finally, it is easy to see that the above two constructions are mutually inverse and extend to an (identity on morphisms) isomorphism between the categories of  $\#$ -algebras and  $\mathbb{T}$ - $\Sigma$ -bialgebras.  $\square$

**Corollary 4.6** *Let  $X \# Y = T(X + Y)$  for a monad  $\mathbb{T}$  on  $\mathbf{C}$ . The category  $\mathbf{C}^{\mathbb{T}}$  of  $\mathbb{T}$ -algebras is isomorphic to the full subcategory of those  $\#$ -algebras  $a : T(A + A) \rightarrow A$ , which factor through  $T\nabla : T(A + A) \rightarrow TA$ .*

Analogously to complete Elgot monads, we introduce  $\#$ -algebras with iteration. This generalizes the definition of a *complete Elgot algebra for a functor* from [5].

**Definition 4.7 (Complete Elgot  $\#$ -algebras)** A *complete Elgot  $\#$ -algebra* is a  $\#$ -algebra  $a : A \# A \rightarrow A$  equipped with an iteration operator

$$\frac{e : X \rightarrow A \# X}{e^\dagger : X \rightarrow A}$$

satisfying the following axioms:

- *solution*: for every  $e : X \rightarrow A \# X$  we have

$$\begin{array}{ccc} X & \xrightarrow{e^\dagger} & A \\ e \downarrow & & \uparrow a \\ A \# X & \xrightarrow{A \# e^\dagger} & A \# A \end{array}$$

- *functoriality*: for every  $e : X \rightarrow A \# X$ ,  $f : Y \rightarrow A \# X$  and  $h : X \rightarrow Y$ ,

$$\begin{array}{ccc} X & \xrightarrow{e} & A \# X \\ h \downarrow & & \downarrow A \# h \\ Y & \xrightarrow{f} & A \# Y \end{array} \quad \text{implies} \quad \begin{array}{ccc} X & & \\ h \downarrow & \searrow e^\dagger & A \\ Y & \nearrow f^\dagger & \end{array}$$

$$f h = (\text{id} \# h) e \text{ implies } f^\dagger h = e^\dagger;$$

- *compositionality*: for every  $f : Y \rightarrow A \# Y$  and  $g : X \rightarrow Y \# X$  define

$$f^\dagger \bullet g = (X \xrightarrow{g} Y \# X \xrightarrow{f^\dagger \# \text{id}} A \# X)$$

and  $f \blacksquare g : Y + X \rightarrow A \# (Y + X)$  by

$$\begin{array}{ccc} Y + X & \xrightarrow{[u_Y^X, g]} & Y \# X \xrightarrow{f \# \text{id}} (A \# Y) \# X \\ & & \downarrow (\text{id} \# \text{inl}) \# \text{inr} \\ A \# (Y + X) & \xleftarrow{m_A^{Y+X}} & (A \# (Y + X)) \# (Y + X) \end{array}$$

Compositionality states that  $(f \blacksquare g)^\dagger \text{inr} = (f^\dagger \bullet g)^\dagger : X \rightarrow A$ .

A *morphism* from a complete Elgot  $\#$ -algebra  $(A, a, -^\dagger)$  to a complete Elgot  $\#$ -algebra  $(B, b, -^\dagger)$  is a  $\mathbf{C}$ -morphism  $f : A \rightarrow B$ , such that for all  $e : X \rightarrow A \# X$  we

have:

$$\left( X \xrightarrow{e^\dagger} A \xrightarrow{f} B \right) = \left( X \xrightarrow{e} A \# X \xrightarrow{f \# \text{id}} B \# X \right)^\dagger.$$

This defines the category of complete Elgot  $\#$ -algebras  $\mathbf{CElgt}_\#(\mathbf{C})$ .

**Remark 4.8** Note that complete Elgot  $\#$ -algebras for the parametrized monad  $A \# X = A + \Sigma X$  (i.e. the parametrized monad of Example 4.3 (i) for  $\mathbb{T}$  the identity monad) are precisely the complete Elgot algebras for the functor  $\Sigma$  introduced and studied in [5].

Like in the case of complete Elgot monads, a standard way to obtain complete Elgot  $\#$ -algebras is by enforcing a suitable enrichment over complete partial orders with bottom.

**Example 4.9 (Continuous algebras are complete Elgot algebras)** Consider any category  $\mathbf{C}$  that is enriched over  $\mathbf{Cppo}$  such that composition is left strict, i.e.  $\perp f = \perp$ , and a parametrized monad  $\# : \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$  that is *locally continuous* in both arguments, i.e.

$$\bigsqcup_i (f_i \# g_i) = \left( \bigsqcup_i f_i \right) \# \left( \bigsqcup_i g_i \right)$$

holds for any  $\omega$ -chains  $(f_i : X \rightarrow Y)_{i < \omega}$  and  $(g_i : X' \rightarrow Y')_{i < \omega}$ . Then every  $\#$ -algebra becomes a complete Elgot  $\#$ -algebra when equipped with the operation  $-^\dagger$  assigning to every  $e : X \rightarrow A \# X$  its least solution. In more detail, let  $A$  be a  $\#$ -algebra. To every  $e : X \rightarrow A \# X$  we assign  $e^\dagger : X \rightarrow A$  given by

$$e^\dagger = \bigsqcup_i e_i^\dagger,$$

where  $e_0^\dagger = \perp : X \rightarrow A$  and  $e_{i+1}^\dagger = a(\text{id} \# e_i^\dagger)e$ . That means that  $e^\dagger$  is the least fixed point of the function  $s \mapsto a(\text{id} \# s)e$  on  $\mathbf{C}(X, A)$ . The verification that this satisfies the axioms of a complete Elgot  $\#$ -algebra can be found in the full version of our paper.

**Example 4.10** The previous example can easily be generalized as follows. Let  $\#$  be a parametrized monad on an arbitrary category  $\mathbf{C}$ . Suppose that  $a : A \# A \rightarrow A$  is a  $\#$ -algebra such that

- (i) for every object  $X$ ,  $\mathbf{C}(X, A)$  is a cpo with  $\perp$ ,
- (ii) for every morphism  $g : X \rightarrow Y$ , the map  $\mathbf{C}(g, A) : \mathbf{C}(Y, A) \rightarrow \mathbf{C}(X, A)$  with  $f \mapsto fg$  is continuous,
- (iii) the map  $f \mapsto a(\text{id} \# f)$  is a continuous map on  $\mathbf{C}(X, A)$ .

Then clearly for every  $e : X \rightarrow A \# X$  the least solution  $e^\dagger$  exists; indeed, the map  $s \mapsto a(\text{id} \# s)e$  is continuous on  $\mathbf{C}(X, A)$ . And the assignment  $e \mapsto e^\dagger$  of a least solution turns  $A$  into a complete Elgot  $\#$ -algebra. The proof of this fact is identical to the proof for the previous example.

Note that if  $\mathbf{C} = \mathbf{Set}$  and  $A$  is a cpo with  $\perp$  then  $\mathbf{C}(X, A)$  is equipped with the pointwise cpo structure and then conditions (i) and (ii) follow automatically.

For illustrative purposes we proceed to describe one concrete instance of this scenario. Let  $X \# Y = X + Y \times Y$  on  $\mathbf{C} = \mathbf{Set}$ . Let  $S$  be a set and let  $S' = S + \{0, \perp\}$ .

Then

$$(S', \text{seq\_or} : S' + S' \times S' \rightarrow S')$$

is a  $\#$ -algebra under the following assignments:

$$\text{seq\_or}(x) = x \quad \text{seq\_or}(\perp, x) = \perp \quad \text{seq\_or}(s, x) = s \quad \text{seq\_or}(0, x) = x$$

where  $s \in S$ . Moreover,  $S'$  is equipped with the flat cpo structure, i.e.  $x \sqsubseteq y$  iff  $x = \perp$  or  $x = y$ , and  $\text{seq\_or}$  is continuous. Then all hom-sets  $\mathbf{C}(X, S')$  are, of course, cpos with  $\perp$  under the pointwise order, and it is then easy to see that our three conditions above are satisfied; for condition (iii) one uses that  $S' + S' \times S'$  is also a cpo (without bottom) and that  $\text{seq\_or}$  is clearly continuous. Thus  $S'$  is a complete Elgot  $\#$ -algebra.

Suppose we have a predicate  $p : S \rightarrow 2$  on  $S$  and a function  $f : X \rightarrow S + X \times X$  representing a graph over the set of nodes  $X$  (where every vertex has either two outgoing transitions or none and is labeled in  $S$ ). Let  $p? : S \rightarrow S'$  be defined by  $p?(s) = s$  if  $p(s) = 1$  and  $p?(s) = 0$  otherwise and consider the map

$$g = \left( X \xrightarrow{f} S + X \times X \xrightarrow{p? + \text{id}} S' + X \times X \right).$$

Since  $g : X \rightarrow S' \# X$ , we obtain the function  $g^\dagger : X \rightarrow S'$ , which performs the *depth-first search* of the first element of  $S$  satisfying  $p$  starting from a given vertex. The results from  $S'$  are to be interpreted as follows:  $s \in S$  is returned if the element is found, 0 if the element is not found,  $\perp$  indicates the divergence.

We revisit these definitions in Example 5.8 after giving a characterization of complete Elgot algebras.

Note that we did not require a morphism of complete Elgot  $\#$ -algebras to be a morphism of  $\#$ -algebras. In fact, this follows automatically.

**Proposition 4.11** *Let  $f : A \rightarrow B$  be a complete Elgot  $\#$ -algebra morphism from  $(A, a, -^\dagger)$  to  $(B, b, -^\dagger)$ . Then  $f$  is a morphism of  $\#$ -algebras.*

**Proof (Sketch).** The idea is to represent  $a$  as a loop terminating after the first iteration and then deduce preservation of  $a$  by  $f$  from preservation of iteration by  $f$  guaranteed by definition. More concretely, we take

$$e = (\text{id} \# \text{inr}) [\text{id}, u_A^A] : (A \# A) + A \rightarrow A \# ((A \# A) + A)$$

and show that  $e^\dagger = [a, \text{id}]$ . The remaining proof amounts to deriving  $b(f \# f) = f a$  from  $f e^\dagger = ((f \# \text{id}) e)^\dagger$ .  $\square$

## 5 Complete Elgot Algebras as Algebras for a Monad

In this section we show that complete Elgot  $\#$ -algebras can be recognized as precisely Eilenberg-Moore algebras of the monad of generalized coalgebraic resumptions on  $\#$ , which we introduce below.

Recall that it was shown by Uustalu [28] that parametrized monads give rise to monads at least in two different ways:

**Proposition 5.1** *Suppose  $\#$  is a parametrized monad on  $\mathbf{C}$  such that the least fixpoint  $\mu\gamma. X \# \gamma$  (the greatest fixpoint  $\nu\gamma. X \# \gamma$ ) exists for every  $X \in |\mathbf{C}|$ . Then  $\mu\gamma.- \# \gamma$  ( $\nu\gamma.- \# \gamma$ ) is the underlying functor of a monad.*

**Remark 5.2** For the parametrized monad  $X \# Y = T(X + \Sigma Y)$  it is well-known that

$$T_\Sigma^\mu X = \mu\gamma. T(X + \Sigma\gamma)$$

is the object mapping of a monad  $\mathbb{T}_\Sigma^\mu$  (in fact,  $\mathbb{T}_\Sigma^\mu$  is the coproduct of the monad  $\mathbb{T}$  and the free monad on  $\Sigma$ , see [17]). In the following we shall mostly be interested in the case where  $\mu$  is replaced by  $\nu$ , i.e. the monad  $\mathbb{T}_\Sigma$  of (★).

It is known that the initial algebra  $\mu\gamma. X \# \gamma$  carries the free  $\#$ -algebra on  $X$ ; conversely, the free  $\#$ -algebra is an initial  $(X \# -)$ -algebra (see [6, Theorem 2.18]). Here we are interested in the final coalgebras  $\nu\gamma. X \# \gamma$ . One of the goals of this section is to establish that the final  $(X \# -)$ -coalgebra carries the free complete Elgot  $\#$ -algebra on  $X$ , and conversely, assuming a free complete Elgot  $\#$ -algebra on  $X$ , its carrier is a final  $(X \# -)$ -coalgebra (see Corollary 5.10).

From now on we assume that the final coalgebras  $\nu\gamma. X \# \gamma$  exist and denote them  $F_\# X$  (standardly omitting the structure morphisms  $\text{out}_X : F_\# X \rightarrow X \# F_\# X$ ). Recall that  $\text{coit } f : X \rightarrow F_\# Y$  is the unique final morphism induced by a coalgebra  $(X, f : X \rightarrow Y \# X)$ . Following [28], in order to introduce and reason about the monad structure of  $F_\#$ , we use a more flexible *primitive corecursion principle*, derived from the standard *coiteration principle* embodied in  $\text{coit}$ .

**Proposition 5.3** ([28]) *For any endofunctor  $F$  with a final coalgebra  $\nu F$ , and any  $f : X \rightarrow F(\nu F + X)$ , there is a unique morphism  $h : X \rightarrow \nu F$  satisfying  $\text{out } h = F[\text{id}, h] f$ .*

The morphism  $h$  in Proposition 5.3 is said to be defined by primitive corecursion. We use primitive corecursion to slightly generalize the  $\text{coit}$  construct in the special case of  $F_\#$ :

**Lemma 5.4** *For any  $e : X \rightarrow B \# X$  and  $f : B \rightarrow A \# F_\# A$ , there is a unique morphism  $h$  satisfying*

$$\begin{array}{ccc} X & \xrightarrow{e} & B \# X \\ h \downarrow & & \downarrow m_A^{f_\# A} (f \# h) \\ F_\# A & \xrightarrow{\text{out}} & A \# F_\# A. \end{array} \quad (1)$$

For any  $e : X \rightarrow B \# X$  and  $f : B \rightarrow A \# F_\# A$  we denote by

$$\text{coit}(e, f) : X \longrightarrow F_\# A$$

the unique  $h$  making diagram (1) commute. Using (1), the monad structure on  $F_\#$  can be given as follows:

$$\begin{aligned} \eta_X^\nu &= \text{out}^{-1} u_X^{f_\# X} = \text{coit } u_X^X \\ f^\star &= \text{coit}((f \# \text{id}) \text{ out}, \text{out}) \end{aligned} \quad \text{where } f : X \rightarrow F_\# Y$$



This also defines  $\mu^\nu = \text{id}^\star = \text{coit}(\text{out}, \text{out})$ . Note that, by Lemma 5.4,  $f^\star$  is the unique morphism satisfying the equation

$$\text{out } f^\star = m_Y^{F_\#^Y} (\text{out } f \# f^\star) \text{ out}. \quad (2)$$

**Lemma 5.5** *Let  $e : X \rightarrow B \# X$  and  $f : B \rightarrow A \# F_\# A$ . Then*

$$\text{coit}(e, f) = (\text{out}^{-1} f)^\star (\text{coit } e).$$

As an easy corollary of Lemma 5.5 we obtain that  $\text{coit } e = \text{coit}(e, u_X^{F_\# X})$ ; indeed, we have

$$\text{coit}(e, u_X^{F_\# X}) = (\text{out}^{-1} u_X^{F_\# X})^\star (\text{coit } e) = (\eta_X^\nu)^\star (\text{coit } e) = \text{coit } e.$$

We state another useful property in the following lemma:

**Lemma 5.6** *Let  $e : X \rightarrow B \# X$  and  $g : B \rightarrow C$ . Then*

$$F_\# g (\text{coit } e) = \text{coit}((g \# \text{id}) e).$$

The following theorem is our first main result. It establishes an equivalence of complete Elgot  $\#$ -algebras and  $F_\#$ -algebras.

**Theorem 5.7** *For any parametrized monad  $\# : \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$ , the Eilenberg-Moore algebras of  $F_\# = \nu\gamma. - \# \gamma$  are exactly the complete Elgot  $\#$ -algebras. More precisely,  $\mathbf{C}^{F_\#}$  and  $\mathbf{CEl}_{\#}(\mathbf{C})$  are isomorphic categories, witnessed by the following construction (in both directions, morphisms are mapped to themselves):*

- $\mathbf{C}^{F_\#} \rightarrow \mathbf{CEl}_{\#}(\mathbf{C})$ : for a  $F_\#$ -algebra  $(A, \chi : F_\# A \rightarrow A)$  we define a  $\#$ -algebra  $(A, \chi \text{out}^{-1}(\text{id} \# \eta^\nu) : A \# A \rightarrow A, -^\dagger)$  with  $e^\dagger = \chi(\text{coit } e) : X \rightarrow A$  for any  $e : X \rightarrow A \# X$ .
- $\mathbf{CEl}_{\#}(\mathbf{C}) \rightarrow \mathbf{C}^{F_\#}$ : for a  $\#$ -algebra  $(A, a : A \# A \rightarrow A, -^\dagger)$  we define an  $F_\#$ -algebra  $(A, \text{out}^\dagger : F_\# A \rightarrow A)$ .

**Proof (Sketch).** For the direction from  $\mathbf{C}^{F_\#}$  to  $\mathbf{CEl}_{\#}(\mathbf{C})$  we have to verify the axioms of complete Elgot  $\#$ -algebras. The hardest case is that of the compositionality identity. We have on the one hand

$$\begin{aligned} (f^\dagger \bullet g)^\dagger &= \chi \text{coit}(f^\dagger \bullet g) \\ &= \chi \text{coit}((\chi(\text{coit } f) \# \text{id}) g) \\ &= \chi \text{coit}((\chi \# \text{id}) ((\text{coit } f) \# \text{id}) g) \\ &= \chi (F_\# \chi) \text{coit}(((\text{coit } f) \# \text{id}) g) && // \text{ Lemma 5.6} \\ &= \chi \mu^\nu \text{coit}(((\text{coit } f) \# \text{id}) g) && // \chi \text{ is an } F_\# \text{-algebra} \\ &= \chi \text{coit}(((\text{coit } f) \# \text{id}) g, \text{out}), && // \text{ Lemma 5.5} \end{aligned}$$

and on the other hand, by definition,

$$(f \blacksquare g)^\dagger \text{inr} = \chi \text{coit}(m_A^{Y+X} (((\text{id} \# \text{inl}) f) \# \text{inr}) [u_Y^X, g]) \text{inr}.$$

Let us denote  $m_A^{Y+X} (((\text{id} \# \text{inl}) f) \# \text{inr}) [u_Y^X, g]$  by  $h$ . By Lemma 5.4, it suffices to show the identity  $\text{out}(\text{coit } h) \text{inr} = m_A^{F_\# A} (\text{out} \# ((\text{coit } h) \text{inr})) (\text{coit } f \# \text{id}) g$ . The

latter is easy to obtain from the auxiliary equation  $(\text{coit } h) \text{ inl} = \text{coit } f$  whose proof is a routine.

For the direction from  $\mathbf{CElg}_\#(\mathbf{C})$  to  $\mathbf{C}^{F_\#}$ , we have to prove the two axioms of Eilenberg-Moore algebras. The harder one is  $\text{out}^\dagger F_\#(\text{out}^\dagger) = \text{out}^\dagger \mu'$  and it is obtained from the instance of compositionality  $(\text{out}^\dagger \blacksquare \text{out})^\dagger \text{inr} = (\text{out}^\dagger \bullet \text{out})^\dagger$  by establishing  $\text{out}^\dagger [\text{id}, \mu'] = (\text{out}^\dagger \blacksquare \text{out})^\dagger$  and  $\text{out}^\dagger F_\#(\text{out}^\dagger) = (\text{out}^\dagger \bullet \text{out})^\dagger$ . Further calculations ensure that the correspondence between  $\mathbf{CElg}_\#(\mathbf{C})$  and  $\mathbf{C}^{F_\#}$  is functorial and moreover an isomorphism.  $\square$

Let us illustrate Theorem 5.7 by revisiting Example 4.10.

**Example 5.8** Recall that we consider the parametrized monad  $X \# Y = X + Y \times Y$  so that  $F_\# X$  is the set of finite and infinite binary trees whose leaves are labeled in  $X$ . The fact that  $(S', \text{seq\_or} : S' + S' \times S' \rightarrow S', -^\dagger)$  is a  $\#$ -algebra means, equivalently, that  $S'$  is an  $F_\#$ -algebra. In particular, the  $F_\#$ -algebra structure is a function  $\text{Seq\_or} : F_\# S' \rightarrow S'$  that transforms a given binary tree over  $S'$  to a single element of  $S'$  calculated by using the depth-first search strategy seeking the first leaf of the given tree that is labeled by  $S$ : In case of success, the answer is in  $S$ , otherwise the answer is either  $0 \in S'$  meaning that no element from  $S$  was found and  $\perp$  in case of divergence (i.e. the procedure arrived in an infinite branch before any element from  $S$  was detected).

Note further that any function  $f : X \rightarrow S + X \times X = S \# X$  represents a graph as explained in Example 4.10. The unique map  $\text{coit } f : X \rightarrow F_\# S$  into the final coalgebra then computes for every node  $x \in X$  its tree unfolding. Now, starting with a predicate  $p : S \rightarrow 2$ , the function  $g^\dagger : X \rightarrow S'$  defined in Example 4.10 is equal to the composition

$$X \xrightarrow{\text{coit } f} F_\# S \xrightarrow{F_\#(p?) } F_\# S' \xrightarrow{\text{Seq\_or}} S'.$$

Indeed, by Theorem 5.7,  $g^\dagger = \text{Seq\_or}(\text{coit } g) = \text{Seq\_or} \text{coit}((p? + \text{id}) f)$  and the rest follows by Lemma 5.6.

We proceed with the goal of showing that the final coalgebras  $\nu\gamma.X \# \gamma$  are, equivalently, the free complete Elgot  $\#$ -algebras. In fact, given a final  $(X \# -)$ -coalgebra  $\text{out}_X : F_\# X \rightarrow X \# F_\# X$ , it follows from (the proof of) Theorem 5.7 that  $F_\# X$  carries the free  $F_\#$ -algebra on  $X$ , and therefore it carries the free complete Elgot  $\#$ -algebra on  $X$  (because the isomorphism of categories preserves freeness of algebras). It is not difficult to work out that the following morphisms

$$F_\# X \# F_\# X \xrightarrow{\text{out}_X \# \text{id}} (X \# F_\# X) \# F_\# X \xrightarrow{m_X^{F_\# X}} X \# F_\# X \xrightarrow{\text{out}_X^{-1}} F_\# X$$

and

$$X \xrightarrow{u_X^{F_\# X}} X \# F_\# X \xrightarrow{\text{out}_X^{-1}} F_\# X$$

form the algebra structure and universal morphism of a free complete Elgot algebra for  $\#$  on  $X$ . The iteration operator on  $F_\# Y$  is obtained as follows. Given  $e : X \rightarrow F_\# Y \# X$  one forms the following coalgebra  $c : F_\# Y + X \rightarrow Y \# (F_\# Y + X)$  for

$Y \# -$ :

$$\begin{array}{ccc}
 F_{\#}Y + X & \xrightarrow{[u_{F_{\#}Y, e}^X]} & F_{\#}Y \# X \xrightarrow{\text{out} \# \text{id}} (Y \# F_{\#}Y) \# X \\
 & & \downarrow (\text{id} \# \text{inl}) \# \text{inr} \\
 Y \# (F_{\#}Y + X) & \xleftarrow{m_Y^{F_{\#}Y + X}} & (Y \# (F_{\#}Y + X)) \# (F_{\#}Y + X)
 \end{array}$$

Then one puts  $e^{\dagger} = (\text{coit } c) \text{ inr}$ .

Conversely, we have the following result.

**Theorem 5.9** *Suppose that  $\varphi_X : FX \# FX \rightarrow FX$  and  $\eta_X : X \rightarrow FX$  form a free complete Elgot  $\#$ -algebra on  $X$ . Then*

$$X \# FX \xrightarrow{\eta_X \# \text{id}} FX \# FX \xrightarrow{\varphi_X} FX$$

*is an isomorphism, and its inverse is the structure of a final  $(X \# -)$ -coalgebra.*

The proof of the above bijective correspondence between final  $(X \# -)$ -coalgebras and free complete Elgot  $\#$ -algebras is a non-trivial generalization of the proof of [5, Theorem 5.4] from complete Elgot algebras for endofunctors to those for parametrized monads; here we have seen one direction of the bijective correspondence as a consequence of Theorem 5.7 while we outline the proof of Theorem 5.9 in the full version of the paper.

**Corollary 5.10** *A free complete Elgot  $\#$ -algebra on  $X$  is equivalently a final coalgebra for  $X \# -$ .*

To conclude the present section, we show that surprisingly, in any free complete Elgot  $\#$ -algebra  $FY$  the iteration operator always assigns a unique solution to any morphism  $e : Y \rightarrow FY \# Y$ .

**Proposition 5.11** *Suppose that  $\varphi_Y : FY \# FY \rightarrow FY$  and  $\eta_Y : Y \rightarrow FY$  form a free complete Elgot  $\#$ -algebra on  $Y$ . Then for every  $e : X \rightarrow FY \# X$ ,  $e^{\dagger} : X \rightarrow FY$  is a unique solution, i.e. a unique morphism satisfying the solution axiom with  $e$ .*

**Proof.** Recall first from Theorem 5.9 that  $FY$  is (equivalently) a final  $(Y \# -)$ -coalgebra with the structure  $t : FY \rightarrow Y \# FY$  obtained as an inverse of

$$Y \# FY \xrightarrow{\eta_Y \# \text{id}} FY \# FY \xrightarrow{\varphi_Y} FY.$$

Let  $e : X \rightarrow FY \# X$  and consider the following  $(Y \# -)$ -coalgebra

$$\begin{array}{c}
 \bar{e} = (FY \# X \xrightarrow{t \# e} (Y \# FY) \# (FY \# X)) \\
 \downarrow (\text{id} \# u_{FY}^X) \# (\text{id} \# \text{id}) \\
 (Y \# (FY \# X)) \# (FY \# X) \xrightarrow{m_Y^{FY \# X}} Y \# (FY \# X).
 \end{array}$$

Now let  $d : X \rightarrow FY$  be any solution of  $e$ , i.e. we have  $d = \varphi_Y(FY \# d)e$ . We will prove below that  $\varphi_Y(FY \# d) : FY \# X \rightarrow FY$  is a coalgebra homomorphism from

$\bar{e}$  to  $t$ . Since  $\bar{e}$  does not depend on the solution  $d$  we then conclude that

$$e^\dagger = \varphi_Y(\text{id} \# e^\dagger)e = \varphi_Y(\text{id} \# d)e = d$$

using finality of  $FY$  in the middle step.

To finish the proof consider the following diagram:

$$\begin{array}{ccccc}
 FY \# X & \xrightarrow{\text{id} \# d} & FY \# FY & \xrightarrow{\varphi_Y} & FY \\
 \downarrow t \# e & & \downarrow t \# \text{id} & & \downarrow t \\
 (Y \# FY) \# (FY \# X) & \xrightarrow{\text{id} \# (\text{id} \# d)} & (Y \# FY) \# (FY \# FY) & \xleftarrow{m_Y^{FY}} & Y \# FY \\
 \downarrow (\text{id} \# u_{FY}^X) \# \text{id} & & \downarrow (\text{id} \# u_{FY}^{FY}) \# \text{id} & \nearrow (\text{id} \# \varphi_Y) \# \varphi_Y & \\
 (Y \# (FY \# X)) \# (FY \# X) & \xrightarrow{(\text{id} \# (\text{id} \# d)) \# (\text{id} \# d)} & (Y \# (FY \# FY)) \# (FY \# FY) & & \\
 \downarrow m_Y^{FY \# X} & & \downarrow m_Y^{FY \# FY} & & \\
 Y \# (FY \# X) & \xrightarrow{\text{id} \# (\text{id} \# d)} & Y \# (FY \# FY) & \xrightarrow{\text{id} \# \varphi_Y} & Y \# FY
 \end{array}$$

Note first that the left-hand edge is  $\bar{e}$ . The upper left-hand square commutes since  $d$  is a solution of  $e$ , for the part below it use that  $(\text{id} \# d)u_{FY}^X = u_{FY}^{FY}$  holds since  $\text{id} \# d$  is a monad morphism, and the lower left- and right-hand part commute by the laws of  $\#$ . That the upper-right hand part commutes follows from the proof of Theorem 5.9, and the remaining little inner part commutes since  $\varphi_Y u_{FY}^{FY} = \text{id}_{FY}$ , which holds because  $\varphi_Y$  is the structure of a  $\#$ -algebra. Hence  $\varphi_Y(\text{id} \# d)$  is a coalgebra homomorphisms as desired, which completes the proof.  $\square$

## 6 Algebras of Complete Elgot Monads

We are now in a position to apply the results on complete Elgot  $\#$ -algebras developed in the previous sections to explore the connection between complete Elgot monads and complete Elgot algebras. Recall that given a monad  $\mathbb{T}$  and an endofunctor  $\Sigma$  over  $\mathbf{C}$ ,  $X \# Y = T(X + \Sigma Y)$  is a parametrized monad and therefore, by Proposition 5.1,  $\mathbb{T}_\Sigma$  given by  $(\star)$  is a monad. We reserve notation  $\mathbb{T}_\nu$  for the special case when  $\Sigma = \text{id}$ :

$$T_\nu X = \nu\gamma.T(X + \gamma).$$

From a computational point of view,  $T_\nu X$  can be considered as a type of processes triggering a computational effect formalized by  $\mathbb{T}$  at each step and eventually

outputting values from  $X$  in case of successful termination. The unary operation captured by  $\Sigma = \text{Id}$  can be understood as *delaying*. This perspective was previously pursued in [15]. Now, if  $\mathbb{T}$  is a complete Elgot monad, or more generally, any monad equipped with an iteration operator, we can define a *collapsing morphism*  $\delta_X : T_\nu X \rightarrow TX$  as follows:

$$\delta_X = \left( T_\nu X \xrightarrow{\text{out}_X} T(X + T_\nu X) \right)^\dagger, \quad (3)$$

which intuitively flattens every possibly infinite sequence of computational steps of  $T_\nu X$  into a single step of  $TX$ . Let us illustrate this with the following toy example.

**Example 6.1** Let  $TX = \mathcal{P}_{\omega_1}(A^* \times X)$  where  $\mathcal{P}_{\omega_1}$  is the countable powerset functor and  $A$  is some fixed alphabet of *actions* like in Example 1.1. We extend  $T$  to a monad  $\mathbb{T}$  by putting

$$\eta_X(x) = \{(\varepsilon, x)\} \quad \text{and} \quad f^*(s \subseteq A^* \times X) = \{(ww', y) \mid (w, x) \in s, (w', y) \in f(x)\},$$

where  $\varepsilon \in A^*$  is the empty word and  $f : X \rightarrow \mathcal{P}_{\omega_1}(A^* \times Y)$ . It is easy to see that  $\mathbb{T}$  is an  $\omega$ -continuous monad (see Example 3.2) and hence a complete Elgot monad with the iteration operator defined using least fixed points. An element of  $TX$  is intuitively a countably branching process, with results in  $X$ , at each step capable of executing a finite series of actions. Now the collapsing morphism (3) for every process  $p \in T_\nu\{\checkmark\}$  calculates the set  $\text{tr}(p) \subseteq A^*$  of all successful traces of  $p$ .

As we will see later (Theorem 6.4 (i)),  $(TX, T_\nu TX \xrightarrow{\delta_{TX}} TTX \xrightarrow{\mu_X} TX)$  is a  $\mathbb{T}_\nu$ -algebra and hence, by Theorem 5.7, a complete Elgot #-algebra. Hence, for a complete Elgot monad  $\mathbb{T}$ , its free algebras are complete Elgot #-algebras. Our next question concerns the converse: Is it possible to equip a given monad  $\mathbb{T}$  with an iteration operator provided that free  $\mathbb{T}$ -algebras are equipped with structures of complete Elgot #-algebras in a coherent way? It turns out that without any further assumptions on the category of complete Elgot #-algebras almost all laws of complete Elgot monads become derivable. More precisely, we introduce the following class of monads.

**Definition 6.2** A monad  $\mathbb{T}$  is called a *weak complete Elgot monad* if it is equipped with an iteration operator  $-^\dagger$  that satisfies fixpoint, naturality, and uniformity axioms and the following identity: for any  $g : X \multimap Y + X$ ,  $f : Y \multimap Z + Y$  we have

$$\left( Y + X \xrightarrow{[\text{inl}, g]} Y + X \xrightarrow{f + \text{id}} Z + Y + X \right)^\dagger \text{inr} = X \xrightarrow{g^\dagger} Y \xrightarrow{f^\dagger} Z. \quad (4)$$

(See Fig. 3 for the pictorial form.)

It is relatively easy to deduce (4) from the codiagonal identity, hence we obtain

**Proposition 6.3** Any complete Elgot monad is a weak complete Elgot monad.

We now can establish a tight connection between weak complete Elgot monads and complete Elgot #-algebras.

**Theorem 6.4** Let  $\mathbb{T}$  be a monad on  $\mathbf{C}$  and let  $X \# Y = T(X + Y)$ .

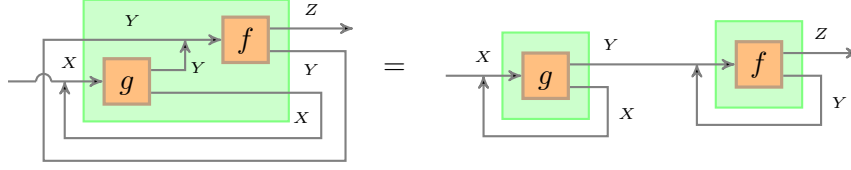


Fig. 3. The additional axiom for weak complete Elgot monads.

- (i) If  $\mathbb{T} = (T, \eta, -^*, -^\dagger)$  is a weak complete Elgot monad then  $\mathbf{C}^\mathbb{T}$  is isomorphic to the full subcategory of  $\mathbf{CElg}_\#(\mathbf{C})$  formed by those complete Elgot  $\#$ -algebras  $(A, a : T(A + A) \rightarrow A, -^\dagger)$  which factor through  $T\nabla : T(A + A) \rightarrow TA$  and for which  $e^\dagger = a(T \text{inl}) e^\dagger$  for every  $e : X \rightarrow T(A + X)$ .
- (ii) Conversely, any functor  $J : \mathbf{C}^\mathbb{T} \rightarrow \mathbf{CElg}_\#(\mathbf{C})$  sending a  $\mathbb{T}$ -algebra  $a : TA \rightarrow A$  to a  $(T\nabla) : T(A + A) \rightarrow A$  and identical on morphisms induces a weak complete Elgot monad structure on  $\mathbb{T}$  as follows:

$$\frac{e : X \rightarrow T(Y + X)}{e^\dagger = (T(\eta + \text{id}) e)^\dagger : X \rightarrow TY} \quad (5)$$

where  $-^\dagger$  is the iteration operator on  $J(TY, \mu)$  (by Clause (i),  $J$  is then full and faithful).

**Remark 6.5** Note that Theorem 6.4(i) can be seen as an analogue of Corollary 4.6 for complete Elgot  $\#$ -algebras.

If  $\mathbf{CElg}_\#(\mathbf{C})$  additionally satisfies a variant of the codiagonal identity, the construction from Clause (ii) of Theorem 6.4 produces precisely complete Elgot monads.

**Theorem 6.6** Let  $\mathbb{T}$  be a monad on  $\mathbf{C}$ , let  $X \# Y = T(X + Y)$  and let  $J : \mathbf{C}^\mathbb{T} \rightarrow \mathbf{CElg}_\#(\mathbf{C})$  be a functor as in Clause (ii) of Theorem 6.4. Then  $\mathbb{T}$  is equipped with the structure of a weak complete Elgot monad given by (5), and moreover  $\mathbb{T}$  is a complete Elgot monad iff every  $(A, a, -^\dagger)$  in the image of  $J$  satisfies the equations

$$(m_{A \# X}^X e)^\dagger = (e^\dagger)^\dagger \quad (6)$$

for every  $e : X \rightarrow (A \# X) \# X$  (this uses the fact that  $A \# X = T(A + X)$  is a free  $\mathbb{T}$ -algebra and hence a complete Elgot  $\#$ -algebra).

## 7 Conclusions and Further Work

We introduced the notion of complete Elgot algebra for a parametrized monad, based on the previous work [4, 28]. We showed that the category of complete Elgot algebras for a parametrized monad  $\#$  is isomorphic to the category of Eilenberg-Moore algebras for the monad  $\nu\gamma. - \# \gamma$  whenever the latter exists. As the category of complete Elgot  $\#$ -algebras is given axiomatically, this can be considered as a form of soundness and completeness result, specifically, it indicates that algebras for  $\nu\gamma. - \# \gamma$  are subject to a lightweight theory of (uniform) iteration.

We explored the connection between complete Elgot  $\#$ -algebras for  $X \# Y = T(X + Y)$  and Eilenberg-Moore algebras of complete Elgot monads, i.e. monads

from [14] supporting a uniform iteration operator satisfying standard axioms of iteration. Specifically, we showed that monads  $\mathbb{T}$  whose algebras are coherently equipped with the structure of a complete Elgot  $\#$ -algebra are precisely complete Elgot monads with the codiagonal axiom replaced by its weakened form (Theorem 6.4). Moreover, if the category of complete Elgot  $\#$ -algebras satisfies a variant of the codiagonal law, such monads  $\mathbb{T}$  are complete Elgot monads (Theorem 6.6).

As an open problem we leave the question whether assumption (6) on complete Elgot algebras in Theorem 6.6 can be lifted. If this was the case, then the notions of weak complete Elgot monads and complete Elgot monads would be equivalent.

We believe that the results we obtained are potentially useful for facilitating constructions over complete Elgot monads, in particular we seek a conceptual simplification for the sophisticated proofs underlying the main result of [14] stating that  $(\star)$  is a complete Elgot monad whenever  $\mathbb{T}$  is. Also we are interested in applications of our results to semantics of abstract side-effecting processes in the style of [15] under equivalences coarser than the behavioral equivalence.

## References

- [1] “Haskell 98 Language and Libraries — The Revised Report,” Cambridge University Press, 2003, also: J. Funct. Prog. **13** (2003).
- [2] Aczel, P., J. Adámek, S. Milius and J. Velebil, *Infinite trees and completely iterative theories: a coalgebraic view*, Theoretical Computer Science **300** (2003), pp. 1–45.
- [3] Adámek, J., R. Börger, S. Milius and J. Velebil, *Iterative algebras: How iterative are they?*, Theory Appl. Cat. **19** (2008), pp. 61–92.
- [4] Adámek, J., S. Milius and J. Velebil, *Iterative algebras for a base*, Electr. Notes Theor. Comput. Sci. **122** (2005), pp. 147–170.
- [5] Adámek, J., S. Milius and J. Velebil, *Elgot algebras*, Log. Methods Comput. Sci. **2** (2006), pp. 1–31.
- [6] Adámek, J., S. Milius and J. Velebil, *Bases for parametrized iterativity*, Inform. and Comput. **206** (2008), pp. 966–1002.
- [7] Adámek, J., S. Milius and J. Velebil, *Equational properties of iterative monads*, Information and Computation **208** (2010), pp. 1306 – 1348.
- [8] Adámek, J., S. Milius and J. Velebil, *Elgot theories: a new perspective of the equational properties of iteration*, Math. Structures Comput. Sci. **21** (2011), pp. 417–480.
- [9] Benton, N. and M. Hyland, *Traced premonoidal categories*, ITA **37** (2003), pp. 273–299.
- [10] Bloom, S. L. and Z. Ésik, “Iteration theories: the equational logic of iterative processes,” Springer-Verlag New York, Inc., New York, NY, USA, 1993.
- [11] Capretta, V., *General recursion via coinductive types*, Logical Methods in Computer Science **1** (2005).
- [12] Elgot, C. C., *Monadic computation and iterative algebraic theories\**, in: H. Rose and J. Shepherdson, editors, *Logic Colloquium ’73 Proceedings of the Logic Colloquium*, Studies in Logic and the Foundations of Mathematics **80**, Elsevier, 1975 pp. 175–230.
- [13] Ésik, Z. and S. Goncharov, *Some remarks on Conway and iteration theories* (2016), arXiv preprint: <http://arxiv.org/abs/1603.00838>.
- [14] Goncharov, S., C. Rauch and L. Schröder, *Unguarded recursion on coinductive resumptions*, in: *Proc. Mathematical Foundations of Programming Semantics XXXI, MFPS 2015, ENTCS*, 2015.
- [15] Goncharov, S. and L. Schröder, *A coinductive calculus for asynchronous side-effecting processes*, Information and Computation **231** (2013), pp. 204 – 232.
- [16] Hasegawa, M., *Recursion from cyclic sharing: Traced monoidal categories and models of cyclic lambda calculi*, in: *Proc. 3rd International Conference on Typed Lambda Calculi and Applications*, Lecture Notes Comput. Sci. **1210** (1997), pp. 196–213.

- [17] Hyland, M., G. Plotkin and J. Power, *Combining effects: Sum and tensor*, Theoret. Comput. Sci. **357** (2006), pp. 70–99.
- [18] Joyal, A., R. Street and D. Verity, *Traced monoidal categories*, Mathematical Proceedings of the Cambridge Philosophical Society **119** (1996), pp. 447–468.
- [19] Kelly, G., *A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on*, Bulletin of the Australian Mathematical Society **22** (1980), pp. 1–83.
- [20] Lawvere, W., *Functorial semantics of algebraic theories*, Proc. Natl. Acad. Sci. USA **50** (1963), pp. 869–872.
- [21] Mac Lane, S., “Categories for the Working Mathematician,” Springer, 1971.
- [22] Moggi, E., *Notions of computation and monads*, Inf. Comput. **93** (1991), pp. 55–92.
- [23] Plotkin, G. and J. Power, *Adequacy for algebraic effects*, in: *FoSSaCS’01*, LNCS **2030**, 2001, pp. 1–24.
- [24] Plotkin, G. and J. Power, *Notions of computation determine monads*, in: *FoSSaCS’02*, LNCS **2303** (2002), pp. 342–356.
- [25] Rutten, J., *Universal coalgebra: a theory of systems*, Technical report, Amsterdam, The Netherlands, The Netherlands (1996).
- [26] Rutten, J. and D. Turi, *Initial algebra and final coalgebra semantics for concurrency*, Springer-Verlag, 1994 pp. 530–582.
- [27] Simpson, A. and G. Plotkin, *Complete axioms for categorical fixed-point operators*, in: *In Proceedings of 15th Annual Symposium on Logic in Computer Science*, 2000, pp. 30–41.
- [28] Uustalu, T., *Generalizing substitution*, ITA **37** (2003), pp. 315–336.



# Classical realizability in the CPS target language

Jonas Frey<sup>1</sup>

*Department of Computer Science  
University of Copenhagen, Denmark  
[jofr@di.ku.dk](mailto:jofr@di.ku.dk)*

---

## Abstract

Motivated by considerations about Krivine’s classical realizability, we introduce a term calculus for an intuitionistic logic with record types, which we call the *CPS target language*. We give a reformulation of the constructions of classical realizability in this language, using the categorical techniques of realizability triposes and toposes.

We argue that the presentation of classical realizability in the CPS target language simplifies calculations in realizability toposes, in particular it admits a nice presentation of conjunction as intersection type which is inspired by Girard’s ludics.

*Keywords:* Classical realizability, ludics, topos, tripos, CPS translation.

---

## 1 Introduction

The relationship between *continuation passing style (CPS) translations* of the  $\lambda$ -calculus, *negative translations* of classical into intuitionistic logic, *control operators* in abstract machines, and *evaluation order* (*call-by-value* vs. *call-by-name*) was uncovered during the 70’s, 80’s, and early 90’s of the past century. The first step was Plotkin [Plo75] recognizing that CPS translations can be used to simulate different evaluation orders within one another. In the 80’s, Felleisen and his collaborators [FFKD86] made the connection between control operators in abstract machines and CPS translations, observing that the behavior of a control operator like `call/cc` in the *source language* of a CPS translation can be implemented by a purely functional expression in the *target language*. Griffin [Gri90] observed the analogy of CPS translations and *negative translations* via the proofs-programs-correspondence, and through this analysis he discovered that the natural type for `call/cc` is *Pierce’s law*, i.e. the propositional schema  $((A \Rightarrow B) \Rightarrow A) \Rightarrow A$ . Since Pierce’s law when added

---

<sup>1</sup> This work is supported by the Danish Council for Independent Research *Sapere Aude* grant “Complexity via Logic and Algebra” (COLA).

to constructive logic yields full classical logic, his observation was celebrated as the unexpected discovery of an *algorithmic meaning of classical logic*.

Negative translations do not require full intuitionistic logic as target logic, and – inspired by Girard’s [Gir91] – Lafont, Reus, and Streicher identified the  $(\neg, \wedge)$ -fragment of intuitionistic logic as sufficient [Laf91, LRS93]. Although in this representation negation is taken as primitive, it is often useful to think of negation as given by the intuitionistic encoding  $\neg A \equiv A \Rightarrow \perp$ , and when constructing models in cartesian closed categories or *response categories* [SR98, Sel01]  $\mathbb{C}$ , one has to interpret  $\perp$  by an object  $R \in \mathbb{C}$  other than the initial object to avoid degeneracy. This  $R$  is called the *response type*, and is comparable to the parameter  $A$  in Friedman’s *A-translation* [Fri78].

Krivine’s *classical realizability* [Kri09] is a realizability interpretation of classical logic which builds on the algorithmic understanding of classical logic arising from Griffin’s insight. It is formulated using an extension of the  $\lambda$ -calculus with *call/cc*, with an operational semantics provided by the *Krivine abstract machine* (KAM) [Kri07]. To interpret logic, the interpretation utilizes a parameter called the *pole*, which plays a role comparable to the response type  $R$ , and to Friedman’s  $A$ , as has been pointed out by Miquel [Miq11].

A motivation of the present work is to make more explicit in which sense the pole plays the role of the response type, by giving a formulation of classical realizability in the *target language* instead of the source language, in which Krivine’s work takes place. To this end, we introduce a term language for a minimal intuitionistic logic based on negation and disjunction (not *conjunction* as Lafont, Reus and Streicher proposed). A design goal is to get a *minimalistic system with a simple operational semantics*, and this is achieved by combining negation and disjunction into a ‘synthetic’ finitary multi-disjunction which should be understood as something like  $\neg(A_1 \vee \dots \vee A_n)$ , but we write as  $\langle \ell_1(A_1), \dots, \ell_n(A_n) \rangle$ , where  $\ell_1, \dots, \ell_n$  are elements of a countable set  $\mathcal{L}$  of *labels*, comparable to *biases* in Girard’s ludics [Gir01]. The CPS target language is a term language of a natural deduction system based on this type constructor scheme.

Instead of presenting the system as a minimal intuitionistic logic based on negation and disjunction, we could also have chosen a presentation as a *dual-intuitionistic* (i.e. using sequents with many formulas on the right and at most one on the left) [Urb96] system based on negation and *conjunction*, which would be closer to Carraro, Salibra, and Ehrhard’s *stack calculus* [CES12], a system which was introduced for similar reasons (as an analysis of Krivine realizability), but is based on *implication* rather than negation. I have chosen the intuitionistic – rather than dual-intuitionistic – presentation for the simple reason that it is easier to handle and does not require as much ‘backward thinking’, but it is good to keep the alternative point of view in mind when comparing with Krivine realizability. In particular, the terms of the CPS target language are *records*, i.e. a kind of tuples, and should be viewed in analogy to *stacks* on the Krivine machine, which fits with the fact that we use sets of *terms* as truth values where Krivine uses sets of *stacks*.

However, we reverse the order on truth values relative to Krivine’s account, and take the empty set as falsity (rather than the set of all stacks as Krivine does), since we use a *call-by-value* translation of classical logic into the target language

instead of the *call-by-name* translation that is implicit in Krivine’s approach. This difference is immaterial from a model-theoretic point of view since it only reverses the order on predicates, which are symmetric as Boolean algebras, but it changes the implementation of classical connectives: where in Krivine realizability, *universal* quantification is the primitive operation that is given by unions of truth values (and the encoding of  $\exists$  is indirect and involves dualization), in our presentation *existential quantification* is the primitive operation. Moreover, in Section 4 we describe how conjunction can be represented as an intersection type under certain (mild) conditions, and together we get a simple representation of the connectives of regular logic (i.e. the  $(\exists, \wedge, \top)$ -fragment of first order logic) not involving the pole at all. This is desirable since regular logic is all that is required for the *tripos-to-topos construction* [HJP80], and a simpler representation of its connectives greatly facilitates calculations in classical realizability toposes.

### 1.1 Related work

The CPS target language is similar in spirit to Thielecke’s *CPS calculus* [Thi97], which can also be motivated as a term calculus for a type system with a kind of multi-negation. The main difference is that in Thielecke’s system the basic type constructor is a negated *n*-ary *conjunction*, and not a negated *n*-ary disjunction as in the CPS target language.

Although different in objective, Curien et al.’s work on term calculi for classical logic [CH00, CMM10] was also inspirational for the present article.

Finally, Terui’s *computational ludics* [Ter11] is a term calculus for ludics designs with a notion of head reduction analogous to the CPS target language. Specifically, the CPS target language can be understood as a non-linear version of the purely additive fragment of the syntax of computational ludics.

## 2 The CPS target language

The syntax of the CPS target language, given in Table 1, distinguishes two syntactic classes called *terms* and *programs*.

A *term* is either a variable or a *record*, i.e. a family  $\langle \ell_1(x_1.p_1), \dots, \ell_n(x_n.p_n) \rangle$  of programs  $p_i$  – the *methods* of the record, each abstracted by a variable  $x_i$  – indexed by a finite subset  $\{\ell_1, \dots, \ell_n\} \subseteq \mathcal{L}$  of a countable set of *labels*, which we take to be the set  $\mathcal{L} = \{\mathbf{a}, \dots, \mathbf{z}\}^*$  of lower case strings (in practice we will only use strings of length 1). The use of different fonts is important: curly  $\ell, \ell$  are placeholders for generic labels, whereas sans-serif  $\mathbf{k}, \mathbf{l}$  are specific labels. The order in which the methods of a record are listed is not important – we view them abstractly as functions from finite sets  $F \subseteq_{\text{fin}} \mathcal{L}$  of labels to programs with a distinguished free variable. In accordance with this viewpoint, we use ‘family notation’  $\langle \ell(x.p) \mid \ell \in F \rangle$  for records when convenient (in particular in Section 4). We refer to the set of labels indexing the methods of a record  $t$  as the *domain* of the record and denote it  $\text{dom}(t)$  – thus  $\text{dom}(\langle \ell_1(x_1.p_1), \dots, \ell_n(x_n.p_n) \rangle) = \{\ell_1, \dots, \ell_n\}$  and  $\text{dom}(\langle \ell(x.p) \mid \ell \in F \rangle) = F$ .

A *program* is an expression of the form  $t_\ell u$ , with the intended meaning that the

<b>Expressions:</b>	
<i>Terms:</i>	$s, t, u ::= x \mid \langle \ell_1(x.p_1), \dots, \ell_n(x.p_n) \rangle$
<i>Programs:</i>	$p, q ::= t_\ell u \mid \dots \text{(possibly non-logical instructions)}$
<b>Reduction:</b>	
	$\langle \ell_1(x.p_1), \dots, \ell_n(x.p_n) \rangle_{\ell_i} t \succ p_i[t/x] \quad \text{if } 1 \leq i \leq n$
<b>Types:</b>	
	$A ::= X \mid \langle \ell_1(A_1), \dots, \ell_n(A_n) \rangle \quad n \geq 0$
<b>Typing rules:</b>	
(Var)	$\frac{}{\Gamma \vdash x_i : A_i} \quad \Gamma \equiv x_1 : A_1, \dots, x_n : A_n, \quad 1 \leq i \leq n$
(Abs)	$\frac{\Gamma, y : B_1 \vdash p_1 \quad \dots \quad \Gamma, y : B_m \vdash p_m}{\Gamma \vdash \langle \ell_1(y.p_1), \dots, \ell_m(y.p_m) \rangle : \langle \ell_1(B_1), \dots, \ell_m(B_m) \rangle}$
(App)	$\frac{\Gamma \vdash t : \langle \ell_1(B_1), \dots, \ell_m(B_m) \rangle \quad \Gamma \vdash u : B_i}{\Gamma \vdash t_{\ell_i} u} \quad 1 \leq i \leq m$

Table 1  
The CPS target language.

program (or *method*) labeled  $\ell$  in  $t$  is called with  $u$  as an argument. This reading suggests the reduction rule  $\langle \ell_1(x_1.p_1), \dots, \ell_n(x_n.p_n) \rangle_{\ell_i} t \succ p_i[t/x_i]$  (provided  $1 \leq i \leq n$ ), which gives the operational semantics of the language. We use the symbol ‘ $\succ$ ’ only for top-level reduction of programs (i.e. *weak head reduction*), and write ‘ $\rightarrow_\beta$ ’ for the compatible closure (i.e. the closure under term and program formers) of  $\succ$  on terms and programs. A *redex* is a program  $t_\ell u$  where  $t$  is a record (not a variable). A redex  $t_\ell u$  with  $\ell \notin \text{dom}(t)$  can not be reduced and is said to be *blocked*. A *normal form* is a term or program that does not contain any redexes, i.e. in every application  $t_\ell u$  the term  $t$  is a variable.

We define the sets  $\text{FV}(t)$  and  $\text{FV}(p)$  of *free variables* of a term or program in the usual way, where the distinguished variable  $x$  of a method  $\ell(x.p)$  in a record  $t$  is considered bound in  $p$ . There are no closed normal programs (since the term in head position cannot be a variable) but there are blocked closed programs like  $\langle \rangle_k \langle \rangle$  and diverging closed programs like  $\langle k(x.x_k x) \rangle_k \langle k(x.x_k x) \rangle$ .

To allow the construction of non-trivial classical realizability models, the syntax has to be extended by non-logical constructs like *constants* or *instructions* to perform side effects<sup>2</sup>. This is achieved by extending the clause for programs in the grammar. To have a model for idealized shell-programs, for example, one can extend the definition of programs to be

$$p, q ::= t_\ell u \mid \mathbf{r}(p, q) \mid \mathbf{w0}(p) \mid \mathbf{w1}(p) \mid \mathbf{0} \mid \mathbf{1}$$

<sup>2</sup> Essentially because of [Fre15b, Lemma 26].

with the intended meaning that the program  $\mathbf{r}(p, q)$  reads a bit from standard input and continues with  $p$  or  $q$  depending on its value,  $\mathbf{w0}(p)$  and  $\mathbf{w1}(p)$  write a 0 or 1, respectively, to standard output before continuing with  $p$ , and  $\mathbf{0}$  and  $\mathbf{1}$  represent successful and unsuccessful termination. For example,  $\langle \mathbf{k}(x. x_k x) \rangle_k \langle \mathbf{k}(x. \mathbf{r}(x_k x, \mathbf{0})) \rangle$  is a program that reads bits from standard input until it encounters a 1, whereupon it terminates successfully.

Formally, such an extension of the syntax has to be accompanied by an extension of the operational semantics, which in the case of the above example can either be given as a labeled transition system or as a transition relation on programs with *state*. This is explained in detail in [Fre15b] using Krivine’s syntax, where it is also explained how in such a setting specifications on program behavior give rise to poles and thus to realizability triposes and toposes. These ideas all transfer to the reformulation of classical realizability given in this article, but instead of formulating our results in this generality – which would require a lot of repetition – we use as running example only a single non-logical constant **end** which represents termination and is comparable to Girard’s *daimon*  $\boxtimes$ <sup>3</sup>. Thus, from now on we assume that programs are of the form

$$p, q ::= t_{\ell u} \mid \mathbf{end}.$$

We denote the sets of closed terms and programs generated by this grammar (together with the rule for terms in Table 1) by  $\mathbb{T}$  and  $\mathbb{P}$ , and more generally we denote by  $\mathbb{T}[x_1, \dots, x_n]$  and  $\mathbb{P}[x_1, \dots, x_n]$  the sets of terms and programs whose free variables are contained in  $\{x_1, \dots, x_n\}$ . The analogous sets of *pure* terms and programs (i.e. those not containing **end**) are denoted by  $\mathbb{T}_0$ ,  $\mathbb{P}_0$ ,  $\mathbb{T}_0[x_1, \dots, x_n]$ , and  $\mathbb{P}_0[x_1, \dots, x_n]$ .

We consider a Curry-style type system for the CPS target language, whose types are generated from type variables and for each finite set  $\{\ell_1, \dots, \ell_n\}$  an  $n$ -ary constructor which forms the record type  $\langle \ell_1(A_1), \dots, \ell_n(A_n) \rangle$  out of types  $A_1, \dots, A_n$ . There are two kinds of typing judgments corresponding to the two syntactic classes:

- *terms*  $t \in \mathbb{T}_0[x_1, \dots, x_n]$  are typed by sequents  $(x_1 : A_1, \dots, x_n : A_n \vdash t : B)$ , and
- *programs*  $p \in \mathbb{P}_0[x_1, \dots, x_n]$  are typed by sequents  $(x_1 : A_1, \dots, x_n : A_n \vdash p)$ .

Thus, programs are not associated to types, but we think of them as having response type (or type  $\perp$ ).

There are three rules (Var), (Abs) and (App), typing *variables*, *records*, and *applications*, respectively. Furthermore, the typing relation is closed under a number of *admissible rules*.

**Lemma 2.1** *The derivable typing judgments are closed under the rules in Table 2.*

**Proof.** Each of the four pairs of rules can be shown to be admissible by simultaneous induction on the structure of  $t$  and  $p$ .  $\square$

A consequence of the admissibility of (Cut) is *subject reduction*.

<sup>3</sup> A referee points out that a concept comparable to the daimon already appears in Coquand’s *evidence semantics* [Coq95].

(Cut)	$\frac{\Gamma \vdash s : A \quad \Gamma, x : A \vdash p}{\Gamma \vdash p[s/x]}$	$\frac{\Gamma \vdash s : A \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash t[s/x] : B}$
(Sym)	$\frac{\Gamma \vdash p}{\sigma(\Gamma) \vdash p}$	$\frac{\Gamma \vdash t : B}{\sigma(\Gamma) \vdash t : B}$
(Weak)	$\frac{\Gamma \vdash p}{\Gamma, x : A \vdash p}$	$\frac{\Gamma \vdash t : B}{\Gamma, x : A \vdash t : B}$
(Contr)	$\frac{\Gamma, x : A, y : A \vdash p}{\Gamma, x : A \vdash p[x/y]}$	$\frac{\Gamma, x : A, y : A \vdash t : B}{\Gamma, x : A \vdash t[x/y] : B}$

Table 2  
Admissible rules for the typing relation, where  $\Gamma \equiv x_1 : A_1, \dots, x_n : A_n$ , and  $\sigma$  is a permutation.

**Lemma 2.2 (Subject reduction)** *If  $\Gamma \vdash \langle \ell_1(x_1.p_1), \dots, \ell_n(x_n.p_n) \rangle_{\ell_i} t$  is derivable for some  $1 \leq i \leq n$ , then  $\Gamma \vdash p_i[t/x_i]$  is derivable.*

**Proof.** Inspection of the typing rules shows that  $\Gamma \vdash \langle \ell_1(x_1.p_1), \dots, \ell_n(x_n.p_n) \rangle_{\ell_i} t$  can only be derived by a deduction

$$\frac{\frac{\Gamma, x_1 : A_1 \vdash p_1 \quad \dots \quad \Gamma, x_n : A_n \vdash p_n}{\Gamma \vdash \langle \ell_1(x_1.p_1), \dots, \ell_n(x_n.p_n) \rangle : \langle \ell_1(A_1), \dots, \ell_n(A_n) \rangle} \quad \Gamma \vdash t : A_i}{\Gamma \vdash \langle \ell_1(x_1.p_1), \dots, \ell_n(x_n.p_n) \rangle_{\ell_i} t}$$

and applying (Cut) to the hypotheses with  $p_i$  and  $t$  yields the claim.  $\square$

### 3 Realizability

Classical realizability models are always defined relative to a *pole*, which is a set  $\perp\!\!\!\perp \subseteq \mathbb{P}$  of closed programs satisfying

$$p \succ q, q \in \perp\!\!\!\perp \Rightarrow p \in \perp\!\!\!\perp \quad (1)$$

for all  $p, q \in \mathbb{P}$ . The deliberations that follow are valid for arbitrary poles satisfying this condition (relative to reasonable extensions of the pure language with non-logical instructions such as in [Fre15b, FS16]), but to have something to hold on to, we fix a pole  $\perp\!\!\!\perp$  by

$$\perp\!\!\!\perp = \{p \mid p \succ^* \mathbf{end}\},$$

which is the set of all programs  $p$  whose weak reduction sequence ‘terminates’, i.e. leads to the constant **end**<sup>4</sup>.

A *truth value* is a set  $S \subseteq \mathbb{T}$  of closed terms. We define as semantic counterparts of the type constructors for each set  $\{\ell_1, \dots, \ell_n\}$  of labels an  $n$ -ary connective on the set  $P(\mathbb{T})$  of truth values.

<sup>4</sup> The classical realizability model arising from this pole has some interesting properties, as the author learned from Krivine [Fre15a].

(Var)	$\frac{}{\Gamma \Vdash x_i : S_i}$	
(App)	$\frac{\Gamma \Vdash t : \langle \ell_1(T_1), \dots, \ell_n(T_n) \rangle \quad \Gamma \Vdash u : T_i}{\Gamma \Vdash t_{\ell_i} u}$	
(Abs)	$\frac{\Gamma, y : T_1 \Vdash p_1 \quad \dots \quad \Gamma, y : T_m \Vdash p_m}{\Gamma \Vdash \langle \ell_1(y.p_1), \dots, \ell_n(y.p_m) \rangle : \langle \ell_1(T_1), \dots, \ell_n(T_m) \rangle}$	
(Cut)	$\frac{\Gamma \Vdash s : S \quad \Gamma, x : S \Vdash p}{\Gamma \Vdash p[s/x]} \quad \frac{\Gamma \Vdash s : S \quad \Gamma, x : S \Vdash t : T}{\Gamma \Vdash t[s/x] : T}$	
(Sym)	$\frac{\Gamma \Vdash p}{\sigma(\Gamma) \Vdash p} \quad \frac{\Gamma \Vdash t : T}{\sigma(\Gamma) \Vdash t : T}$	
(Weak)	$\frac{\Gamma \Vdash p}{\Gamma, x : S \Vdash p} \quad \frac{\Gamma \Vdash t : T}{\Gamma, x : S \Vdash t : T}$	
(Contr)	$\frac{\Gamma, x : S, y : S \Vdash p}{\Gamma, x : S \Vdash p[x/y]} \quad \frac{\Gamma, x : S, y : S \Vdash t : T}{\Gamma, x : S \Vdash t[x/y] : T}$	

Table 3  
Admissible rules for realization judgments, where  $S_1, \dots, S_n, S, T_1, \dots, T_m, T \subseteq \mathbb{T}$ ,  $\Gamma \equiv x_1 : S_1, \dots, x_n : S_n$ , and  $\sigma$  is a permutation.

**Definition 3.1** Given truth values  $S_1, \dots, S_n \in P(\mathbb{T})$  and labels  $\ell_1, \dots, \ell_n \in \mathcal{L}$ , the truth value  $\langle \ell_1(S_1), \dots, \ell_n(S_n) \rangle$  is defined by

$$\langle \ell_1(S_1), \dots, \ell_n(S_n) \rangle = \{t \in \mathbb{T} \mid \forall i \in \{1, \dots, n\} \forall s \in S_i. t_{\ell_i} s \in \perp\}.$$

We introduce *realization judgments* as semantic counterparts of typing judgments.

**Definition 3.2** Given truth values  $S_1, \dots, S_n, T \subseteq \mathbb{T}$ , a term  $t \in \mathbb{T}[x_1, \dots, x_n]$  and a program  $p \in \mathbb{P}[x_1, \dots, x_n]$ ,

$$\text{the notation} \quad x_1 : S_1, \dots, x_n : S_n \Vdash t : T \quad (2)$$

$$\text{stands for} \quad \forall s_1 \in S_1, \dots, s_n \in S_n. t[s_1/x_1, \dots, s_n/x_n] \in T$$

$$\text{and the notation} \quad x_1 : S_1, \dots, x_n : S_n \Vdash p \quad (3)$$

$$\text{stands for} \quad \forall s_1 \in S_1, \dots, s_n \in S_n. p[s_1/x_1, \dots, s_n/x_n] \in \perp.$$

We call expressions of the form (2) and (3) *realization judgments*. Slightly redundantly, we also say ‘the realization judgment  $(\Gamma \Vdash t : T)$  is valid’ instead of simply asserting the judgment itself.

The following result is an analogue of Krivine’s *adequation lemma* [Kri09, Theorem 3].

**Lemma 3.3** *Valid realization judgments are closed under the rules in Table 3.*

**Proof.** The only nontrivial case is (Abs). Assume that  $\Gamma, y : T_k \Vdash p_k$  for  $1 \leq k \leq m$ ,

and that  $s_i \in S_i$  for  $1 \leq i \leq n$ . We have to show that

$$(\langle \ell_1(y.p_1), \dots, \ell_n(y.p_m) \rangle [\vec{s}/\vec{x}])_{\ell_j} t \in \perp\!\!\!\perp$$

for every  $1 \leq j \leq m$  and  $t \in T_j$ . For fixed  $j$  and  $t$  we have

$$\begin{aligned} (\langle \ell_1(y.p_1), \dots, \ell_n(y.p_m) \rangle [\vec{s}/\vec{x}])_{\ell_j} t = \\ (\langle \ell_1(y.p_1[\vec{s}/\vec{x}]), \dots, \ell_n(y.p_m[\vec{s}/\vec{x}]) \rangle)_{\ell_j} t \succ p_j[\vec{s}/\vec{x}, t/y] \end{aligned}$$

where the reduct is in  $\perp\!\!\!\perp$  by assumption, and the claim follows from (1).  $\square$

### 3.1 Classical realizability triposes

We now show how to do classical realizability in the CPS target language by instantiating a simple (call-by-value) negative translation. To start we fix the shorthands

$$\top \equiv \langle \rangle \quad \neg A \equiv \langle \mathbf{k}(A) \rangle \quad \neg(A, B) \equiv \langle \mathbf{l}(A), \mathbf{r}(B) \rangle$$

for nullary, unary, and binary type constructors, and using these we encode classical conjunction as

$$A \wedge B \equiv \neg(\neg A, \neg B). \quad (4)$$

The negative translation maps *classical sequents*

$$A_1, \dots, A_n \vdash B_1, \dots, B_m$$

consisting of formulas built up from propositional variables and the connectives,  $\top$ ,  $\neg$  and  $\wedge$ , to *intuitionistic sequents*

$$A_1^*, \dots, A_n^*, \neg B_1^*, \dots, \neg B_m^* \vdash$$

where the formulas  $A_i^*$  and  $B_j^*$  are obtained by expanding the classical connectives according to the above shorthands and encoding.

We could now define classical realization judgments by mimicking the negative translation on the level of realizability, but we will not spell this out explicitly, and rather develop the remainder of the section in categorical language, by laying out the construction of *classical realizability triposes* analogous to the treatment in [Fre15b].

Broadly speaking, *realizability triposes* [HJP80] capture the model theoretic essence of realizability interpretations as a collection of order relations on sets of *semantic predicates*, which together are required to form an *indexed preorder* – i.e. a contravariant functor  $\mathcal{P} : \mathbf{Set}^{\text{op}} \rightarrow \mathbf{Ord}$  from sets to preorders – subject to certain conditions. The precise definition of *strict Boolean tripos* (which is the version of triposes that we use) is given in Definition A.3.

In our setting, *semantic predicates* on a set  $J$  are functions

$$\varphi, \psi : J \rightarrow P(\mathbb{T})$$



into the set of truth values, and the order on predicates is defined by

$$\varphi \leq \psi \quad :\Leftrightarrow \quad \exists p \in \mathbb{P}_0[x, y] \ \forall j \in J \ . \ (x : \varphi(j), y : \neg\psi(j) \Vdash p), \quad (5)$$

i.e.  $\varphi \leq \psi$  if there exists a *pure* program  $p[x, y]$  which realizes the negative translation of  $\varphi(j) \vdash \psi(j)$  uniformly in  $j$ .

The first step in establishing that semantic predicates form a tripos is to show that the predicates on a fixed set form a *Boolean prealgebra*, i.e. a preorder whose poset reflection is a Boolean algebra (Definition A.1).

**Theorem 3.4** *For every set  $J$ , the set of  $P(\mathbb{T})^J$  of semantic predicates on  $J$  equipped with the order relation (5) is a Boolean prealgebra.*

**Proof.** We show first that  $\leq$  is actually a preorder. Reflexivity follows from the fact that  $(x : S, y : \neg S \Vdash y_k x)$  for arbitrary truth values  $S$ .

For transitivity, assume that  $\varphi \leq \psi$  and  $\psi \leq \theta$ , i.e. that there exist  $p \in \mathbb{P}[v, w]$  and  $q \in \mathbb{P}[x, y]$  such that  $(v : \varphi(j), w : \neg\psi(j) \Vdash p)$  and  $(x : \psi(j), y : \neg\theta(j) \Vdash q)$ .

The claim  $\varphi \leq \theta$  follows from Lemma 3.3 via the derivation

$$\frac{\frac{x : \psi(j), y : \neg\theta(j) \Vdash q}{y : \neg\theta(j) \Vdash \langle k(x.q) \rangle : \neg\psi(j)} \quad v : \varphi(j), w : \neg\psi(j) \Vdash p}{v : \varphi(j), y : \neg\theta(j) \Vdash p[\langle k(x.q) \rangle / w]}$$

Next we show that the order has finite meets. The predicate with value constant  $\top$  is a greatest element, since  $(x : S, y : \neg\top \Vdash y_k \langle \rangle)$  for arbitrary truth values  $S$ . We claim that a binary meet of  $\varphi$  and  $\psi$  is given by pointwise application of (the semantic version of) the type constructor defined in (4), i.e.  $(\varphi \wedge \psi)(j) = \varphi(j) \wedge \psi(j)$ . The such defined  $\varphi \wedge \psi$  is smaller than  $\varphi$  since  $(x : \neg(\neg\varphi(j), \neg\psi(j)), y : \neg\varphi(j) \Vdash x_1 y)$ , and similarly for  $\psi$ . To see that it is a *greatest* lower bound, assume that  $\theta \leq \varphi$  and  $\theta \leq \psi$ , i.e. there exist programs  $p \in \mathbb{P}[w, x]$  and  $q \in \mathbb{P}[w, y]$  such that  $(w : \theta(j), x : \neg\varphi(j) \Vdash p)$  and  $(w : \theta(j), y : \neg\psi(j) \Vdash q)$ . Then we have  $\theta \leq \varphi \wedge \psi$  by the following derivation.

$$\frac{\frac{w : \theta(j), x : \neg\varphi(j) \Vdash p \quad w : \theta(j), y : \neg\psi(j) \Vdash q}{w : \theta(j) \Vdash \langle l(x.p), r(y.q) \rangle : \neg(\neg\varphi(j), \neg\psi(j))}}{w : \theta(j), z : \neg(\neg\varphi(j), \neg\psi(j)) \Vdash z_k \langle l(x.p), r(y.q) \rangle}$$

To finish the proof that  $(P(\mathbb{P})^J, \leq)$  is a Boolean algebra, it now suffices to verify the conditions (i)–(iii) of Lemma A.2, with the negation operation given by  $(\neg\varphi)(j) = \neg\varphi(j)$ .

For (i) assume that  $\varphi \wedge \psi \leq \neg\perp$ , i.e. that there exists  $p[x, y] \in \mathbb{P}[x, y]$  with  $(x : \neg(\neg\varphi(j), \neg\psi(j)), y : \neg\top \Vdash p)$ . Then we have

$$w : \varphi(j), z : \neg\psi(j) \Vdash z_k \langle k(y.p[\langle l(v.v_k w), r(w.w_k y) \rangle / x, \langle k(v.v_k \langle \rangle) \rangle / y]) \rangle$$

(in the following we do not spell out the derivation of realization judgments any more, and leave the type checking to the reader) and hence  $\varphi \leq \neg\psi$ .

For (ii) we have

$$x : \neg(\neg\varphi(j), \neg\neg(\varphi(j))), y : \neg\neg\top \Vdash x_r \langle k(z.x_1z) \rangle$$

and for (iii) we have

$$x : \neg\neg\varphi(j), y : \neg\varphi(j) \Vdash x_k y.$$

□

Every function  $f : J \rightarrow I$  induces a function  $f^* : P(\mathbb{T})^I \rightarrow P(\mathbb{T})^J$  on predicates by precomposition, and it is easy to see that  $f^*$  is monotone and preserves all logical structure (since all propositional operations on predicates are defined pointwise in a uniform way). Since the operation  $(f \mapsto f^*)$  clearly preserves composition and identities, it is the morphism part of a contravariant functor

$$\mathcal{K}_{\perp} : \mathbf{Set}^{\text{op}} \rightarrow \mathbf{BA}.$$

from sets to Boolean prealgebras with object part  $J \mapsto (P(\mathbb{T})^J, \leq)$ . We can now prove the main theorem.

**Theorem 3.5**  $\mathcal{K}_{\perp}$  is a strict Boolean tripos (Definition A.3).

**Proof.** It remains to show that the reindexing maps  $f^*$  admit left adjoints subject to the Beck-Chevalley condition, and that there is a generic predicate.

Let  $f : J \rightarrow I$ . We claim that a left adjoint  $\exists_f$  to  $f^*$  can be defined by fiberwise union, i.e.

$$\exists_f(\varphi)(i) = \bigcup_{fj=i} \varphi(j) \quad \text{for } \varphi \in P(\mathbb{T})^J,$$

and to prove this we have to show that for any  $\psi \in P(\mathbb{T})^I$  we have  $\varphi \leq f^*\psi$  if and only if  $\exists_f\varphi \leq \psi$ . Unfolding definitions yields

$$\exists p \in \mathbb{P}_0[x, y] \forall j \in J \forall s \in \varphi(j) \forall t \in \neg\psi(fj) . p[s, t] \in \perp$$

for the first inequality, and

$$\exists p \in \mathbb{P}_0[x, y] \forall i \in I \forall s \in \bigcup_{fj=i} \varphi(j) \forall t \in \neg\psi(i) . p[s, t] \in \perp$$

for the second one. The two statements are equivalent since in both cases the arguments of  $\varphi$  and  $\psi$  range over all pairs  $(i, j)$  with  $fj = i$ .

It is easy to see (and well known e.g. from the effective tripos) that fiberwise unions strictly satisfy the Beck-Chevalley condition.

Finally, a generic predicate is given by the identity function on  $P(\mathbb{T})$ . □

To conclude the section, we reprove [Fre15b, Lemma 26] in the new syntax.

**Lemma 3.6** *The tripos  $\mathcal{K}_{\perp}$  induced by a pole  $\perp$  is non-degenerate (not equivalent to the terminal tripos) if and only if  $\mathbb{P}_0 \cap \perp = \emptyset$ .*

**Proof.** A tripos is degenerate if and only if all truth values are equivalent, which is easily seen to be equivalent to the existence of a pure program  $p[x] \in \mathbb{P}_0[x]$  such that

the realization judgment  $(x : \mathbb{T} \Vdash p[x])$  holds. If this is the case, then  $p[\langle \rangle] \in \mathbb{P}_0 \cap \perp\!\!\!\perp$ . Conversely, if there exists  $q \in \mathbb{P}_0 \cap \perp\!\!\!\perp$  then we have  $(x : \mathbb{T} \Vdash q)$ .  $\square$

## 4 Conjunction as intersection type

In the previous section we have seen that relative to a fixed pole  $\perp\!\!\!\perp$  the semantic predicates give rise to a tripos  $\mathcal{K}_{\perp\!\!\!\perp}$ , and this tripos in turn gives rise to a *topos*  $\mathbf{Set}[\perp\!\!\!\perp]$  whose construction relies only on the *regular* fragment of first order logic, i.e. the fragment of logic consisting of existential quantification and conjunction. To facilitate computation in classical realizability toposes, it is good to have an easy representations of the basic connectives, and in the proof of Theorem 3.5 we saw that existential quantification in the tripos is given by set theoretic union, which is easy enough. However, for conjunction we only have the representation (4) and the involved double negation entails a high logical complexity and obscures things considerably, i.e. it is difficult to know what the elements of  $S \wedge T$  look like, even if we know the elements of  $S$  and  $T$  very well.

In this section, we show that under certain conditions on the pole we can identify a class of ‘nice’ representatives of predicates in the tripos which admits an implementation of conjunction as intersection type, while being closed under the other logical operations. The idea to represent conjunction as intersection is inspired by ludics [Gir01].

Given a record

$$t = \langle \ell(x.p) \mid \ell \in F \rangle$$

and a set  $M \subseteq \mathcal{L}$  of labels, define the *restriction of  $t$  to  $M$*  to be the record

$$t|_M = \langle \ell(x.p) \mid \ell \in F \cap M \rangle.$$

The *syntactic order*  $\sqsubseteq$  on terms and programs is the reflexive-transitive and compatible (i.e. closed under term and program constructors) closure of the set of all pairs  $(t|_M, t)$  for records  $t$  and sets  $M$  of labels. Observe that the empty record  $\langle \rangle$  is smaller than any other record in the syntactic order, but not smaller than a variable.

**Definition 4.1** A pole  $\perp\!\!\!\perp$  is called *strongly closed*, if it satisfies the conditions

$$\begin{aligned} p \rightarrow_{\beta}^* q, q \in \perp\!\!\!\perp &\Rightarrow p \in \perp\!\!\!\perp \quad \text{and} \\ p \sqsubseteq q, p \in \perp\!\!\!\perp &\Rightarrow q \in \perp\!\!\!\perp, \end{aligned}$$

i.e. it is closed under inverse  $\beta$ -reduction and upward w.r.t. the syntactic order.

A *truth value*  $S \subseteq \mathbb{T}$  is called *strongly closed*, if it satisfies the analogous conditions

$$\begin{aligned} t \rightarrow_{\beta}^* u, u \in S &\Rightarrow t \in S \quad \text{and} \\ t \sqsubseteq u, t \in S &\Rightarrow u \in S. \end{aligned}$$

Although strong closure is a much stronger condition on a pole than mere closure under inverse head reduction, it is satisfied for many ‘reasonable’ poles, in particular for the pole of terminating programs, and more generally for poles constructed from specifications as in [Fre15b].

For a fixed strongly closed  $\perp$ , there is an easy way to strongly close any given truth value, via a well-known double duality construction. Concretely, for  $S \subseteq \mathbb{T}$  define

$$S^\uparrow = \{p[x] \in \mathbb{P}[x] \mid \forall s \in S. p[s] \in \perp\},$$

and dually for  $E \subseteq \mathbb{P}[x]$  define

$$S^\downarrow = \{s \in \mathbb{T} \mid \forall p[x] \in E. p[s] \in \perp\}.$$

If  $\perp$  is strongly closed, it is obvious that so is  $S^{\uparrow\downarrow}$  for any truth value  $S$ .

A truth value  $S$  is said to be *supported* by a set  $M \subseteq \mathcal{L}$  of labels, if we have  $s|_M \in S$  for every  $s \in S$ . More generally, a predicate  $\varphi \in P(\mathbb{T})^J$  is said to be supported by  $M$ , if  $\varphi(j)$  is supported by  $M$  for all  $j \in J$ .

The main result of the section is the following.

**Theorem 4.2** *Let  $\varphi, \psi \in P(\mathbb{T})^J$  be predicates that are both pointwise strongly closed, and supported by disjoint finite sets  $F = \{\ell_1, \dots, \ell_n\}$  and  $G = \{k_1, \dots, k_m\}$  of labels, respectively. Then the predicate  $\varphi \cap \psi$ , which is defined by  $(\varphi \cap \psi)(j) = \varphi(j) \cap \psi(j)$ , is a meet of  $\varphi$  and  $\psi$  and is supported by  $F \cup G$ .*

**Proof.** We claim that the realization judgments

$$x : \varphi(j) \cap \psi(j) \Vdash x : \varphi(j) \qquad x : \varphi(j) \cap \psi(j) \Vdash x : \psi(j)$$

and

$$\begin{aligned} x : \varphi(j), y : \psi(j) &\Vdash u[x, y] : \varphi(j) \cap \psi(j) \\ \text{with } u[x, y] &= \langle \ell_1(z. x_{\ell_1} z), \dots, \ell_n(z. x_{\ell_n} z), k_1(z. y_{k_1} z), \dots, k_m(z. y_{k_m} z) \rangle \end{aligned}$$

hold for all  $j$ . The first two are obvious. For the third one assume that  $s \in \varphi(j)$  and  $t \in \psi(j)$ . Then for each  $\ell_i \in \text{dom}(s)$  the redex  $s_{\ell_i} z$  in  $u[s, t]$  can be reduced, and the result  $u'[s, t]$  satisfies  $u'[s, t] \sqsupseteq s|_F$ . We have  $s|_F \in \varphi(j)$  since  $\varphi(j)$  is supported by  $F$ , and  $u'[s, t] \in \varphi(j) \cap \psi(j)$  and  $u[s, t] \in \varphi(j)$  by strong closure. An analogous argument shows that  $u[s, t]$  is in  $\psi(j)$ , and therefore in  $\varphi(j) \cap \psi(j)$ . The claim that  $\varphi \cap \psi$  is a meet of  $\varphi$  and  $\psi$  now follows from the following lemma.

To see that  $\varphi \cap \psi$  is supported by  $F \cup G$ , assume that  $t \in \varphi(j) \cap \psi(j)$  for some  $j \in J$ . Then  $t|_{F \cup G} \sqsupseteq t|_F \in \varphi(j)$  and by strong closure we have  $t|_{F \cup G} \in \varphi(j)$ .  $\square$

**Lemma 4.3** *If  $\varphi, \psi, \theta \in P(\mathbb{T})^J$  are predicates and  $s[z], t[z] \in \mathbb{T}_0[z]$  and  $u[x, y] \in \mathbb{T}_0[x, y]$  are pure terms such that the realization judgments*

$$z : \theta(j) \Vdash s[z] : \varphi(x) \qquad z : \theta(j) \Vdash t[z] : \psi(x) \qquad x : \varphi(j), y : \psi(j) \Vdash u[x, y] : \theta(x, y)$$

for all  $j \in J$ , then  $\theta$  is a meet of  $\varphi$  and  $\psi$ .

**Proof.** From the first two judgments we can deduce  $(z : \theta(j), v : \neg\varphi(j) \Vdash v_k s[z])$  and  $(z : \theta(j), v : \neg\psi(j) \Vdash v_k t[z])$ , which means that  $\theta \leq \varphi$  and  $\theta \leq \psi$ , and thus  $\theta \leq \varphi \wedge \psi$ . From the third judgment we can derive

$$w : \neg(\neg\varphi(j), \neg\psi(j)), z : \neg\theta(j) \Vdash w_l \langle k(x. w_r \langle k(y. z_k u[x, y]) \rangle) \rangle,$$

which means that  $\varphi \wedge \psi \leq \theta$ . □

Thus we have a nice representation of conjunction for pointwise strongly closed predicates which are finitely supported by disjoint sets.

Disjointness can always be achieved by renaming, i.e. ‘relocating’, as long as supports are *finite*. Moreover, strong closure and finite support are preserved by existential quantification, and by the semantic type constructors (Definition 3.1) provided the the pole is strongly closed. A finitely supported and strongly closed generic predicate can also be obtained, by negating the canonical one given by the identity on  $P(\mathbb{T})$ .

## A Boolean (pre)algebras and Boolean triposes

This appendix recalls the definitions of *Boolean (pre)algebra* and *strict Boolean tripos*, and states an auxiliary lemma to characterize Boolean prealgebras.

**Definition A.1** A *Boolean algebra* is a complemented distributive lattice, i.e. a distributive lattice  $(B, \leq, \top, \wedge, \perp, \vee)$  such that for every  $a \in B$  there exists a  $b \in B$  with  $a \wedge b = \perp$  and  $a \vee b = \top$ .

A *Boolean prealgebra* is a preorder whose poset-reflection is a Boolean algebra.

The term ‘Boolean prealgebra’ does not seem to be very prevalent in the literature, but it appears e.g. in [Fla85].

**Lemma A.2** A preorder  $(B, \leq)$  is a Boolean prealgebra if and only if it has finite meets (denoted by  $\wedge, \top$ ) and there exists a function  $\neg(-) : B \rightarrow B$  such that

$$(i) \ a \wedge b \leq \neg\top \Rightarrow a \leq \neg b \quad (ii) \ a \wedge \neg a \leq \neg\top \quad (iii) \ \neg\neg a \leq a$$

for all  $a, b \in B$ .

**Proof.** The following derivation shows that  $\neg(-)$  is antimonotone.

$$\frac{\frac{a \leq b}{\neg b \wedge a \leq \neg b \wedge b} \quad \frac{}{\neg b \wedge b \leq \neg\top}}{\frac{\neg b \wedge a \leq \neg\top}{\neg b \leq \neg a}}$$

The converse implication of (i) is shown as follows.

$$\frac{\frac{a \leq \neg b}{a \wedge b \leq \neg b \wedge b} \quad \frac{}{\neg b \wedge b \leq \neg\top}}{a \wedge b \leq \neg\top}$$

The following shows that  $\neg(-)$  is an involution,

$$\frac{a \wedge \neg a \leq \neg\top}{a \leq \neg\neg a}$$

which implies that  $(A, \leq)$  is auto-dual and hence a lattice. The non-trivial direction

of distributivity is shown as follows.

$$\begin{array}{c}
\frac{}{\neg(a \wedge b) \wedge a \wedge b \leq \neg\top} \\
\frac{}{\neg(a \wedge b) \wedge a \leq \neg b} \\
\frac{}{\neg(a \wedge b) \wedge a \wedge \neg c \leq \neg b \wedge \neg c} \\
\frac{}{\neg(a \wedge b) \wedge a \wedge \neg c \leq \neg\neg(\neg b \wedge \neg c)} \\
\frac{}{\neg(\neg b \wedge \neg c) \wedge \neg(a \wedge b) \wedge a \wedge \neg c \leq \neg\top} \\
\frac{}{\neg(\neg b \wedge \neg c) \wedge \neg(a \wedge b) \wedge \neg c \leq \neg a} \quad \frac{}{\neg(\neg b \wedge \neg c) \wedge \neg(a \wedge b) \wedge \neg c \leq \neg c} \\
\frac{}{\neg(\neg b \wedge \neg c) \wedge \neg(a \wedge b) \wedge \neg c \leq \neg a \wedge \neg c} \\
\frac{}{\neg(\neg b \wedge \neg c) \wedge \neg(a \wedge b) \wedge \neg c \leq \neg\neg(\neg a \wedge \neg c)} \\
\frac{}{\neg(\neg a \wedge \neg c) \wedge \neg(\neg b \wedge \neg c) \wedge \neg(a \wedge b) \wedge \neg c \leq \neg\top} \\
\frac{}{\neg(\neg a \wedge \neg c) \wedge \neg(\neg b \wedge \neg c) \leq \neg(\neg(a \wedge b) \wedge \neg c)} \\
\hline
(a \vee c) \wedge (b \vee c) \leq (a \wedge b) \vee c
\end{array}$$

It remains to check that for  $a \in A$ ,  $\neg a$  is a complement of  $a$  in the sense of the previous definition. This follows from (ii) and the fact that  $\neg(-)$  is an involution.  $\square$

The following definition of *strict Boolean tripos* is a special case of the concept of tripos as introduced in [HJP80].

**Definition A.3** A *strict Boolean tripos* is a contravariant functor

$$\mathcal{P} : \mathbf{Set}^{\text{op}} \rightarrow \mathbf{BA}$$

from the category of sets to the category of Boolean prealgebras and structure preserving maps such that

- for any  $f : J \rightarrow I$ , the map  $\mathcal{P}(f)$  has a left <sup>5</sup> adjoint  $\exists_f$  (which is not required to preserve Boolean prealgebra structure), such that for any pullback square

$$\begin{array}{ccc}
L & \xrightarrow{q} & K \\
p \downarrow & & \downarrow g \\
J & \xrightarrow{f} & I
\end{array}$$

we have  $\mathcal{P}(g) \circ \exists_f = \exists_q \circ \mathcal{P}(p)$  (this is the *Beck-Chevalley condition*), and

- there exists a *generic predicate*, i.e. a set  $\mathbf{Prop}$  and an element  $\text{tr} \in \mathcal{P}(\mathbf{Prop})$  such that for every set  $I$  and  $\varphi \in \mathcal{P}(I)$  there exists a unique  $f : I \rightarrow \mathbf{Prop}$  with  $\mathcal{P}(f)(\text{tr}) = \varphi$ .

## Acknowledgements

The ideas presented in this article were developed over a long period of time, and I profited from discussions on related issues with many people, including – but not limited to – Pierre Clairambault, Pierre-Louis Curien, Nicolas Guenot, Paul Blain Levy, Paul-André Melliès, Guillaume Munch-Maccagnoni, Jakob Grue Simonsen, Thomas Streicher, Noam Zeilberger, and Stéphane “El Zím” Zimmermann.

Thanks to the referees for their careful rereading and helpful comments.

<sup>5</sup> Note that the right adjoint  $\forall_f$  is for free in the Boolean case, it is given by  $\forall_f \varphi = \neg \exists_f \neg \varphi$ .

## References

- [CES12] A. Carraro, T. Ehrhard, and A. Salibra. The stack calculus. In *Proceedings Seventh Workshop on Logical and Semantic Frameworks, with Applications, LSFA 2012, Rio de Janeiro, Brazil, September 29-30, 2012.*, pages 93–108, 2012.
- [CH00] P.L. Curien and H. Herbelin. The duality of computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000.*, pages 233–243, 2000.
- [CMM10] P.L. Curien and G. Munch-Maccagnoni. The duality of computation under focus. In *Theoretical computer science*, volume 323 of *IFIP Adv. Inf. Commun. Technol.*, pages 165–181. Springer, Berlin, 2010.
- [Coq95] T. Coquand. A semantics of evidence for classical arithmetic. *J. Symbolic Logic*, 60(1):325–337, 1995.
- [FFKD86] M. Felleisen, D. Friedman, E. Kohlbecker, and B. Duba. Reasoning with continuations. In *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*, pages 131–141, 1986.
- [Fla85] R.C. Flagg. Church’s thesis is consistent with epistemic arithmetic. In *Intensional mathematics*, volume 113 of *Stud. Logic Found. Math.*, pages 121–172. North-Holland, Amsterdam, 1985.
- [Fre15a] J. Frey. Computability and Krivine realizability. Note of a conversation with J.L. Krivine, available at <https://sites.google.com/site/jonasfreysite/krivine-comp.pdf>, 2015.
- [Fre15b] J. Frey. Realizability toposes from specifications. In *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland*, pages 196–210, 2015.
- [Fri78] H. Friedman. Classically and intuitionistically provably recursive functions. In *Higher set theory (Proc. Conf., Math. Forschungsinst., Oberwolfach, 1977)*, volume 669 of *Lecture Notes in Math.*, pages 21–27. Springer, Berlin, 1978.
- [FS16] J. Frey and J.G. Simonsen. Toposes for Time Complexity Classes, 2016. *Developments in Implicit Computational Complexity (DICE 2016)*, available at [https://lipn.univ-paris13.fr/DICE2016/Abstracts/paper\\_6.pdf](https://lipn.univ-paris13.fr/DICE2016/Abstracts/paper_6.pdf).
- [Gir91] J.Y. Girard. A new constructive logic: classic logic. *Mathematical Structures in Computer Science*, 1(03):255–296, 1991.
- [Gir01] J.Y. Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(03):301–506, 2001.
- [Gri90] T. Griffin. A formulae-as-type notion of control. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 47–58. ACM, 1990.
- [HJP80] J.M.E. Hyland, P.T. Johnstone, and A.M. Pitts. Tripos theory. *Math. Proc. Cambridge Philos. Soc.*, 88(2):205–231, 1980.
- [Kri07] J.L. Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.
- [Kri09] J.L. Krivine. Realizability in classical logic. *Panoramas et synthèses*, 27:197–229, 2009.
- [Laf91] Y. Lafont. Negation versus implication. *Logical Frameworks*, pages 223–229, 1991.
- [LRS93] Y. Lafont, B. Reus, and T. Streicher. Continuation semantics or expressing implication by negation. Unpublished, available at <http://iml.univ-mrs.fr/~lafont/pub/continuation.ps>, 1993.
- [Miq11] A. Miquel. Existential witness extraction in classical realizability and via a negative translation. *Log. Methods Comput. Sci.*, 7(2):2:2, 47, 2011.
- [Plo75] G.D. Plotkin. Call-by-name, call-by-value and the  $\lambda$ -calculus. *Theoret. Comput. Sci.*, 1(2):125–159, 1975.
- [Sel01] P. Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Math. Structures Comput. Sci.*, 11(2):207–260, 2001.
- [SR98] T. Streicher and B. Reus. Classical logic, continuation semantics and abstract machines. *Journal of functional programming*, 8(06):543–572, 1998.
- [Ter11] K. Terui. Computational ludics. *Theor. Comput. Sci.*, 412(20):2048–2071, 2011.
- [Thi97] H. Thielecke. Continuation semantics and self-adjointness. *Electronic Notes in Theoretical Computer Science*, 6:348–364, 1997.
- [Urb96] I. Urbas. Dual-intuitionistic logic. *Notre Dame J. Formal Logic*, 37(3):440–451, 1996.

# Categorical Models of the Differential $\lambda$ -Calculus Revisited

J.R.B. Cockett<sup>1,2</sup>

*Department of Computer Science  
University of Calgary  
Calgary, Canada*

J.D. Gallagher<sup>3</sup>

*Department of Computer Science  
University of Calgary  
Calgary, Canada*

---

## Abstract

The paper shows that the Scott-Koymans theorem for the untyped  $\lambda$ -calculus extends to the differential  $\lambda$ -calculus. The main result is that every model of the untyped differential  $\lambda$ -calculus may be viewed as a differential reflexive object in a Cartesian closed differential category. This extension of the Scott-Koymans theorem depends critically on unravelling the somewhat subtle issue of which idempotents can be split so that differential structure lifts to the idempotent splitting.

The paper uses (total) Turing categories with “canonical codes” as the basic categorical semantics for the  $\lambda$ -calculus. It shows how the main result may be developed in a modular fashion by first adding left-additive structure to a Turing category, and then – on top of that – differential structure. For both levels of structure it is necessary to identify how “canonical codes” behave with respect to the added structure and, furthermore, how “universal objects” behave. The latter is closely tied to the question – which is the crux of the paper – of which idempotents can be split in these more structured settings.

*Keywords:* Scott-Koymans, Differential Lambda Calculus, Categorical Models

---

## 1 Introduction

In [12], Ehrhard and Regnier introduced the differential  $\lambda$ -calculus to give a syntactic counterpart for the models of linear logic which Ehrhard had introduced in [10,11]. In these models proofs were interpreted as differentiable maps with the linear maps, in the sense of linear logic, becoming rather elegantly the maps

---

<sup>1</sup> partially supported by NSERC

<sup>2</sup> Email: [robin@cpsc.ucalgary.ca](mailto:robin@cpsc.ucalgary.ca)

<sup>3</sup> Email: [jdgallag@ucalgary.ca](mailto:jdgallag@ucalgary.ca)



which were linear in the differential sense. The differential  $\lambda$ -calculus, introduced a new aspect because, as described in [12], it was an untyped system with a confluent rewriting system and, thus provided a model of computability. Furthermore, it was immediately apparent that this calculus was very closely related to the resource  $\lambda$ -calculus [6,5,7]. This confluence of developments, thus, initiated the study of models of computability in which the computable functions admit a differentiation.

The categorical semantics for these settings was developed in series of papers. Initially the tensorial (or linear logic) side of the story was developed in [2]: this closely followed the path of Ehrhard's more model driven development. Tensor differential categories (also written  $\otimes$ -differential categories) came equipped with a comonad (an exponential modality)<sup>4</sup> and it was understood that the differential  $\lambda$ -calculus would then be interpreted in the coKleisli category for this comonad.

Of course, this rather indirect approach did not facilitate the exploration of these coKleisli categories which rapidly become the main focus of attention. The next step – motivated not least by desire to understand the axiomatic behaviour of differentiation in classical calculus – was, therefore, to develop a direct axiomatization for these categories. Toward this end Cartesian differential categories (also written  $\times$ -differential categories) were introduced [3]. Importantly, these were more general than simply being an axiomatization of the coKleisli category for a  $\otimes$ -differential category. While it is certainly the case that coKleisli categories of  $\otimes$ -differential categories are  $\times$ -differential categories<sup>5</sup>, the converse is certainly not true.

The next step in this development involved  $\times$ -differential categories which were – in the appropriate sense – Cartesian closed. In [8] a sound and complete interpretation of the simply typed differential  $\lambda$ -calculus into Cartesian closed differential categories was provided. Furthermore, the connection between the resource  $\lambda$ -calculus and the differential  $\lambda$ -calculus was then implicitly determined by tying both to the same categorical semantics. To complete the story it thus only remained to provide a precise account of the semantics of the untyped differential  $\lambda$ -calculus. At this stage, it was abundantly clear that the interpretation of the untyped differential calculus should be into some sort of reflexive object in a Cartesian closed differential category and, thus, there should be an analogue of the Scott-Koymans<sup>6</sup> theorem [14,18] which says that all model arise from such a situation.

In [17], Manzonetto initiated the investigation of models of the untyped differential  $\lambda$ -calculus. He showed that linear reflexive objects in a Cartesian differential

<sup>4</sup> In fact,  $\otimes$ -differential categories come with different strengths of axiomatizations. In [2] an effort was made to obtain the weakest possible axiomatization. Of note, however, is the stronger notion which was introduced by Marcello Fiore in [13].

<sup>5</sup> A precise characterization of when a Cartesian differential category is the coKleisli category of a  $\otimes$ -differential category is described in [4].

<sup>6</sup> At the time of the development of this theorem, Koymans was a PhD. student, Dana Scott was widely publicizing his results, and also Jim Lambek with Phil Scott were in the process of writing their book, [15], and, indeed, were circulating chapters for comment. Their book, in particular, has a section on C-monoids which provide the semantics for the untyped  $\lambda$ -calculus with both  $\eta$ - and  $\beta$ -equality. In addition, they introduced the notion of a “weak” C-monoid in the exercises and noted that these provide the semantics of the  $\lambda$ -calculus with just  $\beta$ -equality. The notion of “weak” there is exactly our notion of having canonical codes. Note that, in this paper, we use “weak” to mean that the choice *does not* necessarily admit substitution.

category soundly interpret the untyped differential  $\lambda$ -calculus. Furthermore, he gave a completeness theorem, for the differential  $\lambda$ -calculus, however, with two additional equations:

$$\lambda x.(Ds \cdot t)x = Ds \cdot t \quad \text{and} \quad a + a = a.$$

Thus, he did not manage to provide a general completeness theorem which paralleled the Scott-Koymans result. In this paper we revisit this problem and we show that by subtly changing the requirements on the idempotents which one splits one can indeed obtain a complete analogue of the Scott-Koymans result. In Manzonetto's work it was assumed – very reasonably – that the idempotents one should split had to be linear. Unfortunately, this meant that, as Curry's pairing combinator,  $(a, b) \mapsto \lambda p.p a b$ , is not linear one is forced to look for an alternative idempotent which can encode pairing. Manzonetto actually found such a combinator: it was the more complicated  $(a, b) \mapsto \lambda y.(a + Dy \cdot b)$ . However, in order to make this combinator behave correctly, he had to restrict to the case where addition was idempotent.

To access the more general situation, which is the subject of this paper, it is necessary to understand more fully the behaviour of idempotents in a Cartesian differential category. To achieve this one not only has to understand their behaviour with respect to differentiation but also with respect to the left-additive structure. We shall, therefore, redevelop this program by describing the categorical semantics not only of models of the differential  $\lambda$ -calculus but also of models of the “additive”  $\lambda$ -calculus. This is because moving directly from a Cartesian closed differential setting to a model of the differential  $\lambda$ -calculus, without this intermediate step, tends to obscure the subtle behaviour required of the idempotents which must be split. In fact, in this paper we shall start the development even further back: we actually redevelop the Scott-Koymans theorem itself from the more modern and unifying perspective of Turing categories [9]. One justification for this is that, in the theory of Turing categories, idempotents and idempotent completions play a very central role. This predisposes one to take the behaviour of idempotents very seriously as one begins to add structure.

Because our aim is to model the  $\lambda$ -calculus, we shall exclusively focus on Turing categories all of whose maps are total: these were introduced by Longo and Moggi [16] – albeit under a different name – and their realizability theory was developed by Birkedal [1]. Such a Turing category is a weakly Cartesian closed category with an object,  $T$ , called a Turing object which is universal. An object in a category is *universal* in case all objects are a retracts of it. In order to model the untyped  $\lambda$ -calculus, with  $\beta$ -equality, we shall employ the notion of a Turing category with canonical codes. This allows us to interpret the term calculus of the untyped  $\lambda$ -calculus very directly into these categorial models. In particular, this step does not involve consideration of idempotents. This approach, in fact, closely parallels the approach Koymans followed in [14], and allows a convenient separation of concerns which is useful later. To recapture the Scott-Koymans theorem one observes that splitting idempotents yields a Cartesian closed category in which the Turing object is still universal: this immediately makes that the Turing object a reflexive object.

This is the approach that we then follow, in a modular fashion, as we successively add structure. First we add left-additive structure and develop the categorical semantics of the untyped additive  $\lambda$ -calculus, and then, following the same pattern, arrive at the categorical semantics of the untyped differential  $\lambda$ -calculus. At each stage we must appropriately strengthen both the notion of being weakly cartesian closed (with canonical codes) and of being a universal object – where the notion of a universal object, as one adds structure, crucially depends on understanding which idempotents can be split.

The payoff of this reconstruction is that it allows a general analogue for the Scott-Koymans theorem for untyped differential  $\lambda$ -models.

## 2 The $\lambda$ -Calculus

The classical notion of a categorical model of the untyped  $\lambda$ -calculus is as a reflexive object in a cartesian closed category. This is an object  $U$  which has as a retract its object of endomorphisms,  $[U, U] \triangleleft_r^s U$ .<sup>7</sup> One way to approach the Scott-Koymans completeness result is to interpret the  $\lambda$ -calculus across this retraction: this is rather messy as one must repeatedly use the section and retraction to interpret terms. An alternative approach, which parallels the approach of Koymans, is to interpret the  $\lambda$ -calculus into a Turing category with canonical codes and then show that splitting the idempotents results in a Cartesian closed category in which the Turing object is still universal. This approach gives an elegant separation of the two concerns (interpretation and idempotent splitting) and, furthermore, allows the introduction of a more direct categorical counterpart to the untyped  $\lambda$ -calculus: namely Turing categories with canonical codes.

### 2.1 Syntax of the $\lambda$ -calculus

The syntax of the  $\lambda$ -calculus may be described as “untyped” terms in context. The term formation rules consist of term formation rules for a Cartesian theory, see table 1, and the special terms for the “untyped”  $\lambda$ -calculus see Table 2.

In the term formation rules for a Cartesian theory we insist that the variables of a context are distinct. We shall only allow variables for atomic types and allow the formation of variable patterns corresponding to composite Cartesian products of objects. This allows a smooth transition to the categorical semantics as we can turn any sequent into a map, a sequent with one premise:

$$p : X \vdash t : T$$

by simply collecting the context, using the pattern rules, into one type. When substituting for a pattern one then has to match the term being substituted to the pattern: thus, for example, the substitution  $[(t, s)/(x, y)]$  becomes, equivalently,  $[t/x, s/y]$ . The type system ensures that term and pattern in a substitution have exactly the same form.

<sup>7</sup> Here  $[U, U]$  is the internal hom sometimes written  $U^U$  or  $U \Rightarrow U$ . The notation  $[U, U] \triangleleft_r^s U$  means  $[U, U]$  is a retract of  $U$  with  $s : [U, U] \rightarrow U$  the section and  $r : U \rightarrow [U, U]$  the retraction so  $sr = 1_{[U, U]}$ .

$\frac{A \text{ an atomic type}}{\Gamma, x : A, \Gamma' \vdash x : A} \text{ Projection}$	
$\frac{\Gamma, \Gamma' \vdash t : X}{\Gamma, () : 1, \Gamma' \vdash t : X} \text{ Unit Pattern}$	$\frac{\Gamma, x : X, y : Y, \Gamma' \vdash t : Z}{\Gamma, (x, y) : X \times Y, \Gamma' \vdash t : Z} \text{ Pair Pattern}$
$\frac{\Gamma \vdash t : X \quad \Gamma \vdash s : Y}{\Gamma \vdash (t, s) : X \times Y} \text{ Pairing}$	$\frac{\Gamma, \Gamma' \vdash vs : X \quad \Gamma, p : X, \Gamma' \vdash t : Z}{\Gamma, \Gamma' \vdash t[s/p] : Z} \text{ Cut/Substitution}$

Table 1  
Term formation rules for a Cartesian theory

$\frac{\Gamma \vdash m : U \quad \Gamma \vdash n : U}{\Gamma \vdash mn : U} \text{ Application}$	$\frac{\Gamma, x : U, \Gamma' \vdash m : U}{\Gamma, \Gamma' \vdash \lambda x. m : U} \text{ Abstraction}$
--	---

Table 2  
Term formation rules for the  $\lambda$ -calculus

As usual we allow  $\alpha$ -conversion of bound variables and  $\beta$ -equality

$$\lambda x. m \equiv_{\alpha} \lambda y. (m[y/x]) \quad (\lambda x. m)n \equiv_{\beta} m[n/x]$$

and generate the theory  $\Lambda_{\beta}$  by forming the smallest congruence on terms that contains the above equations. There is an associated category of this theory,  $\mathcal{C}(\Lambda_{\beta})$ , called the classifying category of  $\Lambda_{\beta}$ :

**Objects:** Words in  $U, \times, 1$ .

**Arrows:** A map  $T \rightarrow S$  corresponds to a sequent with one type on the left hand side

$$p : T \vdash m : S$$

under the term equivalence of  $\Lambda_{\beta}$ .

**Composition:** Composition is substitution:

$$\frac{p : X \vdash m : Y \quad p' : Y \vdash n : Z}{p : X \vdash n[m/p'] : Z}$$

We note, rather modestly, that this is a Cartesian category:

**Proposition 2.1**  $\mathcal{C}(\Lambda_{\beta})$  is a Cartesian category.

The rest of this section develops the properties of this category.

## 2.2 Total Turing categories with canonical codes

A **(total) Turing category**  $\mathbb{X}$  is a Cartesian category with:

[T.1] An object  $T$  and for each pair of objects  $B, C$  a map  $T \times B \xrightarrow{\bullet_{BC}} C$ ;

[T.2] For each map  $A \times B \xrightarrow{f} C$  a map  $A \xrightarrow{c_f} T$  such that

$$\begin{array}{ccc} T \times B & \xrightarrow{\bullet_{BC}} & C \\ c_f \times 1 \uparrow & \nearrow f & \\ A \times B & & \end{array}$$

A Turing category has **canonical codes** when there is a function

$$\mathbb{X}(A \times B, C) \xrightarrow{\lambda(\cdot)} \mathbb{X}(A, T)$$

such that

$$\begin{array}{ccc} & T & \\ \lambda[(g \times 1)f] \nearrow & \uparrow \lambda[f] & \\ D \xrightarrow{g} & A & \end{array} \quad \begin{array}{ccc} T \times B & \xrightarrow{\bullet_{BC}} & C \\ \lambda[f] \times 1 \uparrow & \nearrow f & \\ A \times B & & \end{array}$$

commute. This means that this function,  $\lambda[\cdot]$ , for forming codes has a substitutional property, namely that  $\lambda[(g \times 1)f] = g\lambda[f]$  and that  $\lambda[f]$  is indeed a code for  $f$  in the sense that  $(\lambda(f) \times 1) \bullet_{BC} = f$ .

Observe first that the property of having canonical codes can be simplified:

**Lemma 2.2** *A Turing category  $\mathbb{X}$  has canonical codes if and only if there is a code  $\lambda[\bullet_{BC}]$  for each  $\bullet_{BC} : T \times B \rightarrow C$  such that whenever  $(c_1 \times 1) \bullet_{BC} = f = (c_2 \times 1) \bullet_{BC}$  that is  $c_1$  and  $c_2$  are codes for  $f$ , then  $c_1 \lambda[\bullet_{BC}] = c_2 \lambda[\bullet_{BC}]$ .*

**Proof.** The proof follows from considering the following diagram

$$\begin{array}{ccc} T \times B & \xrightarrow{\bullet} & C \\ \lambda(\bullet) \times 1 \uparrow & \nearrow \bullet & \\ T \times B & & \\ c_i \times 1 \uparrow & \nearrow f & \\ A \times B & & \end{array}$$

$\lambda(f) \times 1$  (curved arrow from  $A \times B$  to  $T \times B$ )

□

The following is an economical “recognition” theorem for Turing categories with canonical codes:

**Lemma 2.3** *To have a Turing category with canonical codes is to have a Cartesian category with a universal object  $T$  and a “Turing” map  $T \times T \xrightarrow{\bullet} T$  which has canonical codes for all maps  $T \times T \rightarrow T$ .*

**Proof.** For the universality of the Turing object, in a Turing category, obtain the section as a code for  $A \times 1 \xrightarrow{\pi_0} A$ . To extract a Turing structure from a universality of  $T$  and a Turing map, derive a universal application for arbitrary objects by

$$T \times B \xrightarrow{1 \times s_B} T \times T \xrightarrow{\bullet} T \xrightarrow{r_C} C.$$

The canonical code for  $f$  is  $\lambda[f] := s_A \lambda[(r_A \times r_B) f s_C]$ .  $\square$

We use this to show:

**Theorem 2.4**  $\mathcal{C}(\Lambda_\beta)$  is a Turing category with canonical codes.

**Proof.** The universal map is  $U \times U \xrightarrow{\bullet} U; (m, n) \vdash mn$ . By lemma 2.3, it suffices to show that  $\bullet$  has canonical codes for terms  $U \times U \rightarrow U; (x, y) \vdash t$ . Define  $\lambda[(x, y) \vdash t] := x \vdash \lambda y. t$ . It is easy to see that this is a canonical code.

1 is a retract of  $U$ . To see that  $U \times U$  is a retract of  $U$  use Curry's pairing and projection combinators: the section is  $(a, b) \vdash \lambda p. p a b$ , and for the retraction take  $a \vdash (a (\lambda xy. x), a (\lambda xy. y))$ .  $\square$

From a logical perspective this theorem proves the *completeness* of the untyped  $\lambda$ -calculus with respect to models which are Turing categories with canonical codes. With some more work (some of which is developed below) one can exhibit this as part of an adjunction between  $\lambda$ -theories and Turing categories with canonical codes. As this is off the path of our development we shall leave it for a fuller exposition.

### 2.3 Interpreting the $\lambda$ -calculus

We now show that Turing categories with canonical codes are sound models for the untyped  $\lambda$ -calculus by showing there is a canonical functor  $\llbracket \_ \rrbracket : \mathcal{C}(\Lambda_\beta) \rightarrow \mathbb{X}$  which carries the universal object onto the Turing object and application onto the Turing morphism  $\bullet : T \times T \rightarrow T$ . At this stage we should make an important remark: a Turing category may have more than one possible Turing object and more than one Turing morphism as one only needs the existence of such structure to be a Turing category. Thus, in this development, we need to strengthen the notion of a Turing category to specify, as part of being a Turing category, the intended Turing structure. In particular, for a Turing category with canonical codes the function which supplies the canonical codes will be part of this structure.

With this understanding, let  $\mathbb{X}$  be a Turing category with canonical codes, whose Turing object is  $T$  and universal map is  $T \times T \xrightarrow{\bullet} T$ . We define the interpretation of the untyped  $\lambda$ -calculus,  $\llbracket \_ \rrbracket : \mathcal{C}(\Lambda_\beta) \rightarrow \mathbb{X}$ , as follows:

**Objects:** On objects define  $\llbracket X \rrbracket$  to be  $X[T/U]$ .

**Maps:** On arrows:

$$\begin{aligned}
 \llbracket p : X \vdash () : 1 \rrbracket &= !\llbracket X \rrbracket \\
 \llbracket p : X \vdash (m, n) : Y \times Z \rrbracket &= \langle \llbracket p : X \vdash m : Y \rrbracket, \llbracket p : X \vdash n : Z \rrbracket \rangle \\
 \llbracket x : U \vdash x : U \rrbracket &= 1_T \\
 \llbracket (p_1, p_2) : X_1 \times X_2 \vdash x : U \rrbracket &= \pi_i \llbracket p_i : X_i \vdash x : U \rrbracket \quad \text{where } x \in p_i \\
 \llbracket p : X \vdash mn : U \rrbracket &= \langle \llbracket p : X \vdash m : U \rrbracket, \llbracket p : X \vdash n : U \rrbracket \rangle \bullet \\
 \llbracket p : X \vdash \lambda z. m : U \rrbracket &= \lambda [\llbracket (p, z) : X \times U \vdash m : U \rrbracket]
 \end{aligned}$$

We then have:

**Theorem 2.5** *If  $\mathbb{X}$  is a Turing category with canonical codes,*

$$\llbracket - \rrbracket : \mathcal{C}(\Lambda) \rightarrow \mathbb{X}$$

*is a Cartesian functor which preserves the Turing object, the Turing map, and the canonical codes.*

The proof is standard.

#### 2.4 Reflexive objects

An important observation for Turing categories is that the idempotent splitting of a Turing category  $\mathbb{X}$ ,  $\text{Split}(\mathbb{X})$ , is again a Turing category (see [9]), furthermore, the Turing object and the Turing morphism are preserved in this splitting. Our next objective is to show that splitting the idempotents of a Turing category which has canonical codes results not just in a Turing category with canonical codes but in a Cartesian closed Turing category. As the Turing object is still universal in the idempotent splitting, this, in particular, means that it is a reflexive object. This allows us to conclude that all models of the untyped  $\lambda$ -calculus arise from a reflexive object  $[T, T] \triangleleft_r^s T$  in a Cartesian closed Turing category, and this is how we interpret the Scott-Koymans theorem.

**Theorem 2.6** *When  $\mathbb{X}$  is a Turing category with canonical codes then  $\text{Split}(\mathbb{X})$  is a cartesian closed Turing category.*

The proof is that of Koymans [14].

Note that  $[1_T, 1_T] = \lambda[\bullet]$  is an idempotent which in  $\text{Split}(\mathbb{X})$  witnesses that the Turing object in the splitting is a reflexive object.

### 3 The additive $\lambda$ -calculus

The previous section sets the broad outline for this and the next section. In these sections we will follow our program of adding structure step by step to the untyped  $\lambda$ -calculus. The first step in this program is to consider left-additive structure: it is required in order to discuss differential structure.

A **left-additive category**  $\mathbb{X}$  is a category in which each homset  $\mathbb{X}(A, B)$  is a commutative monoid, and where

$$f(g + h) = fg + fh \quad f0 = 0.$$

$$\frac{\Gamma \vdash m : U \quad \Gamma \vdash n : U}{\Gamma \vdash (m + n) : U} \text{ Addition} \qquad \frac{}{\Gamma \vdash 0 : U} \text{ Zero}$$

Table 3  
Term formation rules addition in the additive  $\lambda$ -calculus

A map  $f$  in a left additive category is **additive** when  $(g + h)f = gf + hf$  and  $0f = 0$ .

A **Cartesian left additive category** is a left additive category with finite products such that

$$(f + g) \times (h + k) = (f \times h) + (g \times k) \qquad 0 \times 0 = 0$$

and where  $\Delta : A \rightarrow A \times A$  and all projections  $\pi_i$  are additive. An important characterization of Cartesian left-additive categories is:

**Proposition 3.1 (Proposition 1.2.2 [3])** *To have a Cartesian left additive category is to have a Cartesian category in which each object,  $X$ , has a canonical commutative monoid structure,  $X \times X \xrightarrow{+x} X \xleftarrow{0x} 1$ , that satisfies the following coherence:*

$$+_{A \times B} := (A \times B) \times (A \times B) \xrightarrow{\text{ex}} (A \times A) \times (B \times B) \xrightarrow{+A \times +B} A \times B.$$

### 3.1 Syntax of the additive $\lambda$ -calculus

The syntax of the untyped additive  $\lambda$ -calculus is again defined by untyped terms in context. The term formation rules use the usual Cartesian rules Table 1, the formation rules for the  $\lambda$ -calculus Table 2, and the additive rules of Table 3.

The equations for  $\beta$ -equality and  $\alpha$ -conversion hold together with equations to make  $(U, +, 0)$  into a commutative monoid, and

$$\begin{aligned} \lambda x.(m + n) &= \lambda x.m + \lambda x.n & \text{and} & & \lambda x.0 &= 0 \\ (m + n)a &= ma + na & \text{and} & & 0a &= 0. \end{aligned}$$

These equations generate the theory  $\Lambda_{\beta+}$ .

As before there is an associated category,  $\mathcal{C}(\Lambda_{\beta+})$ , called the classifying of the additive  $\lambda$ -calculus. The objects are words in  $U, \times, 1$  and the arrows are sequents with one premise on the left of the turnstile. Composition is, again, substitution. We state modestly:

**Proposition 3.2**  $\mathcal{C}(\Lambda_{\beta+})$  is a Cartesian category in which  $U$  is a commutative monoid.

Our next task is to develop the categorical semantics of the additive  $\lambda$ -calculus, and thus, the properties of this category. To do this we must address two related issues: namely what it means to be an *additive* universal object and what it means to have *additive* canonical codes.



### 3.2 Additive universal objects

The idea behind a universal object in a category is that the entire category is determined by the monoid of endomorphisms of that object. For plain categories this just means that every object is a retract of  $U$ . However, in a Cartesian left additive category a universal object must have a stronger property as it must also induce the additive structure on each object. An object  $U$  in a left-additive category  $\mathbb{X}$  is an **additive universal** object in case every object,  $A$ , is a retract of  $U$  in such a manner that the retraction  $r_A$ , in  $A \triangleleft_{r_A}^{s_A} U$ , is an additive map. This requirement ensures that the additive structure on  $A$  is determined by that of  $U$ :

**Lemma 3.3** *In any left-additive Cartesian category with a retract  $A \triangleleft_{r_A}^{s_A} U$  which has  $r_A$  additive:*

- (i)  $A \times A \xrightarrow{+_A} A = A \times A \xrightarrow{s_A \times s_A} U \times U \xrightarrow{+} U \xrightarrow{r_A} A$  and  $0_A = 0r_A$ ;
- (ii)  $e_A = r_A s_A$  satisfies  $+_U e_A = (e_A \times e_A) +_U e_A$ .

Note that (i) implies that  $(fe + ge)r = (f + g)r$ . When an idempotent  $e : U \rightarrow U$  satisfies part (ii), that is  $+_U e = (e \times e) +_U e$ , we will say  $e$  is a **retractively additive** idempotent.

The following – completely general – lemma lets us define additive structure on objects using additive retractions:

**Lemma 3.4** *If  $(U, +_U, 0_U)$  is a commutative monoid in a Cartesian category,  $\mathbb{X}$ , and  $A \triangleleft_{r_A}^{s_A} U$  with  $e_A = r_A s_A$  retractively additive – that is  $(e_A \times e_A) +_U e_A = +_U e_A$  – then there is a unique commutative monoid structure on  $A$  which makes  $r_A$  a homomorphism.*

This means, when  $U$  is a universal object, which is a commutative monoid, we may, using Lemma 3.4, induce additive structure on any retract  $A$  of  $U$ , whose induced idempotent is retractively additive: this will automatically make the retract additive. In order, to create left-additive structure on the whole category from the additive structure on  $U$ , we must select a particular way in which each object is an additive retract so that we may induce a unique additive structure on each object. Furthermore, to ensure the result is a *Cartesian* left-additive category, it is also necessary for these induced additions to be compatible with the product: this means we must demand that the additive structure on the product  $A \times B$  be defined componentwise.

A **universal structure**,  $\mathcal{U}$ , for  $U \in \mathbb{X}$  consists of, for each  $A \in \mathbb{X}$ , a way in which  $A$  is a retract of  $U$ ,  $\mathcal{U}(A) = A \triangleleft_{r_A}^{s_A} U$ . Clearly if  $U$  has a universal structure it must be a universal object. A Cartesian category,  $\mathbb{X}$ , has **chosen products** in case there are chosen functors  $\times : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{X}$  and  $1 : \mathbb{1} \rightarrow \mathbb{X}$  right adjoint to the diagonal and final functors respectively. In a Cartesian category with chosen products, if  $(U, +_U, 0_U)$  is a commutative monoid, then a universal structure  $\mathcal{U}$  for  $U$  is said to be **additively coherent** in case:

$$[\text{UAC.1}] \quad \mathcal{U}(U) = U \triangleleft_{1_U}^{1_U} U;$$

[UAC.2] The retraction,  $r_{U \times U}$ , of  $\mathcal{U}(U \times U)$  satisfies:

$$\begin{array}{ccc} U \times U & \xrightarrow{+_U} & U \\ r_{U \times U} \times r_{U \times U} \downarrow & & \downarrow r_{U \times U} \\ (U \times U) \times (U \times U) & \xrightarrow{\text{ex}} (U \times U) \times (U \times U) \xrightarrow{+_U \times +_U} & U \times U \end{array}$$

[UAC.3] Each idempotent  $e_A = r_A s_A$ , of  $\mathcal{U}(A)$  is retractively  $U$ -additive;

[UAC.4]  $\mathcal{U}(A \times B)$  has  $s_{A \times B} = (s_C \times s_B) s_{U \times U}$  and  $r_{A \times B} = r_{U \times U} (r_A \times r_B)$ .

In a left-additive Cartesian category with an additive universal object, there is no guarantee that one has an additively coherent universal structure. However, given an additive universal structure one can generate a left-additive category:

**Proposition 3.5** *If  $\mathbb{X}$  is a category with chosen products and a commutative monoid  $(U, +_U, 0_U)$  which has an additively coherent universal structure,  $\mathcal{U}$ , then there is a unique left additive structure on  $\mathbb{X}$  making each retraction in  $\mathcal{U}$  additive.*

The proof uses the fact that the retractions are homomorphisms of the commutative monoid structure on objects.

### 3.3 Left-additive Turing categories

A **left-additive Turing category** is a Turing category which is Cartesian left-additive and has each universal application  $\bullet_{BC} : T \times B \rightarrow C$  additive in its first argument:

$$\langle h + k, g \rangle \bullet = \langle h, g \rangle \bullet + \langle k, g \rangle \bullet \quad \text{and} \quad \langle 0, g \rangle \bullet = 0$$

A left-additive Turing category  $\mathbb{X}$  has **additive canonical codes** when, as a Turing category,  $\mathbb{X}$  has canonical codes such that, in addition,  $\lambda[f + g] = \lambda[f] + \lambda[g]$  and  $\lambda[0] = 0$ .

We have the following recognition theorem for left-additive Turing categories:

**Proposition 3.6**  *$\mathbb{X}$  is a left-additive Turing category if and only if  $\mathbb{X}$  is Cartesian left-additive, with an additively universal object  $T$  which has a Turing map  $\bullet : T \times T \rightarrow T$  which is additive in its first argument.*

*Furthermore, the Turing category has additively canonical codes if and only if the Turing map has additively canonical codes.*

That objects have additive retractions from  $T$  uses the retraction from Proposition 2.3.

To see that we can construct canonical codes for  $A \times B \rightarrow C$  from canonical codes for  $T \times T \rightarrow T$  does require a bit of care concerning the definition of the addition on the maps: as only  $T$  has additive structure,  $f + g : A \rightarrow B$  should be thought of as  $(fs + gs)r$ , where  $B \triangleleft_s^r T$ . The canonical code is  $s_A \lambda[(r_A \times r_B)(fs_C + gs_C)]$ .

We are now ready to state:

**Theorem 3.7**  *$\mathcal{C}(\Lambda_{\beta+})$  is a left-additive Turing category with additively canonical codes.*

The proof of this theorem relies heavily on the fact that Curry's retraction  $U \rightarrow U \times U$  is additive, and this is enough to show that  $U$  has an additive universal structure. The rest of the proof follows easily.

### 3.4 Interpreting the additive $\lambda$ -calculus

Our next objective is to show, that for any left-additive Turing category,  $\mathbb{X}$ , with additively canonical codes, there is a left-additive functor  $\mathcal{C}(\Lambda_{\beta+}) \rightarrow \mathbb{X}$ . From a logical perspective this says that left-additive Turing categories with additively canonical codes are sound models of the additive  $\lambda$ -calculus.

Let  $\mathbb{X}$  be a left-additive Turing category with additive canonical codes with Turing object  $T$  and Turing map is  $\bullet : T \times T \rightarrow T$ . The functor  $\llbracket - \rrbracket : \mathcal{C}(\Lambda_{\beta+}) \rightarrow \mathbb{X}$  is defined in the same way as before on variables, application, abstraction, and tuples: all we have to describe is the interpretation of the additive structure:

$$\begin{aligned} \llbracket p : X \vdash 0 : U \rrbracket &= 0 : \llbracket X \rrbracket \rightarrow T \\ \llbracket p : X \vdash m + n : U \rrbracket &= \llbracket p : X \vdash m : U \rrbracket + \llbracket p : X \vdash n : U \rrbracket : \llbracket X \rrbracket \rightarrow T \end{aligned}$$

**Proposition 3.8** *When  $\mathbb{X}$  is a left-additive Turing category with additive canonical codes*

$$\llbracket - \rrbracket : \mathcal{C}(\Lambda_{\beta+}) \rightarrow \mathbb{X}$$

*is a Cartesian left-additive functor which preserve the Turing object, the Turing map, and its canonical codes.*

The proof is by calculation and is relatively standard.

Again with more work this interpretation can be turned into an adjunction between additive  $\lambda$ -theories and left-additive Turing categories with additive canonical codes.

### 3.5 Additive reflexive objects

We now wish to split the idempotents of a left-additive Turing category with additive canonical codes. However, it is clear that we cannot split any old idempotent if we want to be able to induce an additive structure on the splitting. So clearly we should split the retractively additive idempotents. However, when we restrict the idempotents we split we have to ensure that all the structure we require, namely the Cartesian left additive structure, is still present.

Clearly all identity maps are additively retractive idempotents. Note that the product of retractively additive idempotents is retractively additive, and the requirements of being retractively additive on an idempotent is exactly what is required to obtain additive structure on each idempotent in the splitting.

Finally, note that the internal homsets of 2.6,  $[u, v] = \lambda[(1 \times u) \bullet_{AB} v]$ , are additively retractive.

A **Cartesian closed left-additive category** is a Cartesian left additive category which is closed and has every evaluation map additive in its first argument. As a Cartesian closed category always has canonical codes, this may equivalently be stated as the requirement that it has additive canonical codes. This gives:

**Theorem 3.9** *If  $\mathbb{X}$  is a left-additive Turing category with additive canonical codes then splitting the additively retractive idempotents,  $\text{Split}_+(\mathbb{X})$  yields a Cartesian closed left-additive Turing category.*

By construction all the additively retractive idempotents split with additive retractions; hence, the retraction  $T \rightarrow [T, T]$  is additive. This immediately means that every model of the additive  $\lambda$ -calculus can be seen to arise as an additive reflexive object in a Cartesian closed left-additive category.

## 4 The Differential $\lambda$ -calculus

A **Cartesian differential category**  $\mathbb{X}$  is a Cartesian left additive category with a combinator

$$\frac{A \xrightarrow{f} B}{A \times A \xrightarrow{D[f]} B}$$

that satisfies seven axioms:

$$[\text{CD.1}] \quad D[0] = 0 \text{ and } D[f + g] = D[f] + D[g]$$

$$[\text{CD.2}] \quad \langle 0, g \rangle D[f] = 0 \text{ and } \langle h + k, g \rangle D[f] = \langle h, g \rangle D[f] + \langle k, g \rangle D[f]$$

$$[\text{CD.3}] \quad D[1] = \pi_0, D[\pi_0] = \pi_0 \pi_0, \text{ and } D[\pi_1] = \pi_0 \pi_1$$

$$[\text{CD.4}] \quad D[\langle f, g \rangle] = \langle D[f], D[g] \rangle$$

$$[\text{CD.5}] \quad D[f g] = \langle D[f], \pi_1 f \rangle D[g]$$

$$[\text{CD.6}] \quad \langle \langle g, 0 \rangle, \langle h, k \rangle \rangle D[D[f]] = \langle g, k \rangle D[f]$$

$$[\text{CD.7}] \quad \langle \langle p, h \rangle, \langle g, k \rangle \rangle D[D[f]] = \langle \langle p, g \rangle, \langle h, k \rangle \rangle D[D[f]]$$

In a Cartesian differential category a map  $f$  is **linear** when  $D[f] = \pi_0 f$ . Intuitively this means

$$\frac{df(x)}{dx}(x) \cdot v = f(v)$$

The following is corollary 2.2.3 in [3]

**Proposition 4.1** *If a map is linear, then it is additive. Moreover, the class of linear maps form a commutative monoid enriched category with biproducts.*

The following lemma is quite useful in many calculations

**Lemma 4.2** *Let  $h$  and  $k$  be linear and  $f$  any map, then*

$$D[hfk] = (h \times h)D[f]k.$$

In a Cartesian differential category, if  $f : A \times B \rightarrow C$ , we may define the **partial derivative** of  $f$  with respect to  $A$  as

$$D_{\times,0} := A \times (A \times B) \xrightarrow{(1,0) \times 1} (A \times B) \times (A \times B) \xrightarrow{D[f]} C$$

$\frac{\Gamma, p : S, \Gamma' \vdash m : U \quad \Gamma, \Gamma' \vdash a : S \quad \Gamma, \Gamma' \vdash v : S}{\Gamma, \Gamma' \vdash \frac{dm}{dp}(a) \cdot v : U}$
---

Table 4  
Term formation rules for the differential  $\lambda$ -calculus

One may obtain the partial derivative of  $f$  with respect to  $B$  by

$$D_{\times,1} := B \times (A \times B) \xrightarrow{\langle 0,1 \rangle \times 1} (A \times B) \times (A \times B) \xrightarrow{D[f]} C$$

We will also make use of maps that are **linear in the first argument**. This means

$$(v, (x, y)) \mapsto \frac{df(x, y)}{dx}(x) \cdot v = f(v, y)$$

i.e. that the partial derivative in the first argument is linear.

The following is useful in many calculations:

**Lemma 4.3** *Suppose  $h : A \times B \rightarrow C$  is linear in its first argument. Then  $(1 \times g)h$  is linear in its first argument for any  $g$ .*

Intuitively this is so because  $g$  does not “touch” the first argument of  $h$ .

#### 4.1 Syntax of the differential $\lambda$ -calculus

The syntax of the untyped differential  $\lambda$ -calculus is again defined by untyped terms in context. Term formation uses the rules of Cartesian theories Table 1, the rules for the  $\lambda$ -calculus Table 2, the rules for the additive  $\lambda$ -calculus, Table 3, and a single new rule for the differential  $\lambda$ -calculus, Table 4. Note that the syntax we use here is slightly different from (but equivalent to) the syntax used by Ehrhard and Regnier in [12]<sup>8</sup>.

We have the following equations on terms

$$[\text{Dt.1}] \quad \frac{dm_1+m_2}{dp}(a) \cdot v = \frac{dm_1}{dp}(a) \cdot v + \frac{dm_2}{dp}(a) \cdot v \quad \frac{d0}{dp}(a) \cdot v = 0$$

$$[\text{Dt.2}] \quad \frac{dm}{dx}(a) \cdot (v_1 + v_2) = \frac{dm}{dx}(a) \cdot v_1 + \frac{dm}{dx}(a) \cdot v_2 \quad \frac{dm}{dx}(a) \cdot 0 = 0$$

$$[\text{Dt.3}] \quad \bullet \quad \frac{dx}{dx}(a) \cdot v = v, \\ \bullet \quad \frac{dt}{d(p,p')}\left((a, a')\right) \cdot (v, v') = \frac{dt[a'/p']}{dp}(a) \cdot v + \frac{dt[a/p]}{dp'}(a') \cdot v'$$

$$[\text{Dt.4}] \quad \frac{dm[t/q]}{dp}(a) \cdot v = \frac{dm}{dq}(t[a/p]) \cdot \frac{dt}{dp}(a) \cdot v$$

Note in the above, by the context-formation rules,  $p \notin \text{fv}(m)$ .

$$[\text{Dt.5}] \quad \frac{d \frac{dm}{dp}(a) \cdot q}{dq}(b) \cdot v = \frac{dm}{dp}(a) \cdot v$$

$$[\text{Dt.6}] \quad \frac{d \frac{dm}{dp_1}(a_1) \cdot v_1}{dp_2}(a_2) \cdot v_2 = \frac{d \frac{dm}{dp_2}(a_2) \cdot v_2}{dp_1}(a_1) \cdot v_1$$

<sup>8</sup> In Ehrhard and Regnier’s syntax,  $D(m) \cdot v := \lambda a. \frac{dm}{dz}(a) \cdot v$ .

**[Dt.7]**  $\frac{d\lambda y.m}{dp}(a) \cdot v = \lambda y. \frac{dm}{dp}(a) \cdot v$   
 By the context formation rules,  $x \notin (a, v)$

**[Dt.8]**  $\frac{d\lambda z.yz}{dy}(a) \cdot v = \lambda z.vz$

Note that when we write  $\frac{dt}{dp}(a) \cdot v$  in **[Dt.4]** the  $t$  could be a tuple of terms. However, this is only a notation for distributing the derivative down onto term of type  $U$  as only these have derivatives. This means  $\frac{d()}{dp}(a) \cdot v = ()$  and  $\frac{d(t_1, t_2)}{dp}(a) \cdot v = (\frac{dt_1}{dp}(a) \cdot v, \frac{dt_2}{dp}(a) \cdot v)$ .

**Lemma 4.4** In  $\Lambda_{\beta\partial}$ :

- (i) If  $y \notin \text{fv}(m)$  then  $\frac{dm}{dy}(a) \cdot v = 0$ ;
- (ii)  $\frac{dm}{d(p, p')}(a, a') \cdot (v, 0) = \frac{dm[a'/p']}{dp}(a) \cdot v$ ;
- (iii)  $\frac{dm}{d(p, p')}(s, s') \cdot (0, v') = \frac{dm[a/p]}{dp'}(a') \cdot v'$ ;
- (iv) If  $y \notin \text{fv}(m)$  then  $\frac{dy m}{dy}(a) \cdot v = vm$ ;
- (v)  $\frac{dm n}{dx}(a) \cdot v = (\frac{dm}{dx}(a) \cdot v)(n[a/x]) + \frac{d(m[a/x])}{dz_1} z_1 (n[a/x]) \cdot \frac{dn}{dx}(a) \cdot v$ .

The proof is straightfoward; the first three are from [3].

There is an associated category  $\mathcal{C}(\Lambda_{\beta\partial})$ . We have

**Proposition 4.5**  $\mathcal{C}(\Lambda_{\beta\partial})$  is a Cartesian category in which  $(U, +_U, 0_U)$  is a commutative monoid.

## 4.2 Differential universal objects

In subsection 3.2, we introduced the notion of an additive universal object that induces an additive structure on each object. In a Cartesian differential category, we must again strengthen the notion of universal object, so that the derivative on maps  $U \rightarrow U$  induces a derivative on all maps. An object  $U$  in a Cartesian differential category  $\mathbb{X}$  is a **differential universal object** in case every object  $A$  is a retract of  $U$ ,  $A \triangleleft_{r_A}^{s_A} U$ , and the retraction is linear.

**Lemma 4.6** In any Cartesian differential category with a retract  $A \triangleleft_{r_A}^{s_A} U$  in which  $r_A$  is linear

- (i) For any  $A \xrightarrow{f} B$ ,  $D[f] = (s_A \times s_A)D[r_A f s_B]r_B$ ;
- (ii)  $e_A = r_A s_A$  satisfies  $(e_A \times e_A)D[e_A] = D[e_A]$  and  $D[e_A]e_A = \pi_0 e_A$

The proof makes repeated use of 4.2.

When an idempotent satisfies part (ii) of the above and is retractively additive, we will say that it is a **retractively linear idempotent**.

We now extend the extend the notion of additively coherent universal structure. In a Cartesian category with chosen products, where  $(U, +_U, 0_U)$  is a commutative monoid, a universal structure  $\mathcal{U}$  for  $U$  is **differentially coherent** if it is additively coherent and additionally:

[UDC.1] There is a differential operator for  $U$ ; i.e.

$$\frac{U \xrightarrow{f} U}{U \times U \xrightarrow{D[f]} U}$$

that satisfies [CD.1,2,3,5].

[UDC.2] Each idempotent  $e_A = r_A s_A$  of  $\mathcal{U}(A)$  is retractively  $U$ -linear in the sense that

$$(e_A \times e_A)D[e_A] = D[e_A] \quad \text{and} \quad D[e_A]e_A = \pi_0 e_A$$

[UDC.3] For  $\mathcal{U}(U \times U) = U \times U \triangleleft_{r_{U \times U}}^{s_{U \times U}}$ , we have that

$$\begin{aligned} (s_{U \times U} \times s_{U \times U})D[r_{U \times U} \pi_0] &= \pi_0 \pi_0 \\ (s_{U \times U} \times s_{U \times U})D[r_{U \times U} \pi_1] &= \pi_0 \pi_1 \end{aligned}$$

And that for any  $f, g : U \rightarrow U$ :

$$D[\langle f, g \rangle_{s_{U \times U}}]r_{U \times U} = \langle D[f], D[g] \rangle$$

[UDC.4] Let  $U \xrightarrow{f} U$ . The map

$$D^2[f] := (U \times U) \times (U \times U) \xrightarrow{s_{U \times U} \times s_{U \times U}} U \times U \xrightarrow{D[r_{U \times U} D[f]]} U$$

satisfies

$$\begin{aligned} \langle \langle g, 0 \rangle, \langle h, k \rangle \rangle D^2[f] &= \langle g, k \rangle D[f] \\ \langle \langle 0, h \rangle, \langle g, k \rangle \rangle D^2[f] &= \langle \langle 0, g \rangle, \langle h, k \rangle \rangle D^2[f] \\ ((e_A \times e_A) \times (e_A \times e_A))D^2[f] &= D^2[f] \end{aligned}$$

Given a differentially coherent universal structure on a category, one can generate a Cartesian differential category.

**Proposition 4.7** *If  $\mathbb{X}$  is a category with chosen products and a commutative monoid  $(U, +_U, 0_U)$  which has a differentially coherent universal structure,  $\mathcal{U}$ , then there is a Cartesian differential structure on  $\mathbb{X}$  that makes each retraction in  $\mathcal{U}$  linear.*

We first show that  $\mathbb{X}$  has a Cartesian differential structure.

Note the following facts about  $\mathcal{U}$  idempotents.

- For all  $U \xrightarrow{f} U$ ,  $D[fe_B]r_B = D[f]r_B$
- For all  $A \xrightarrow{f} B$ ,  $(e_A \times e_A)D[r_A f s_B] = D[r_A f s_B]$

The differential operator  $\mathcal{D}[\_]$  is defined on maps  $A \rightarrow B$ :

$$\frac{A \xrightarrow{f} B}{A \times A \xrightarrow{\mathcal{D}[f]} } = \frac{\frac{A \xrightarrow{f} B}{U \xrightarrow{r_A} A \xrightarrow{f} B \xrightarrow{s_B} U}}{U \times U \xrightarrow{D[r_A f s_B]} U} \quad \frac{A \times A \xrightarrow{s_A \times s_A} U \times U \xrightarrow{D[r_A f s_B]} U \xrightarrow{r_B} B}$$

### 4.3 Differential Turing categories

A **differential Turing category** is a Turing category which is also a Cartesian differential category and where additionally each universal application  $T \times B \xrightarrow{\bullet_{BC}} C$  is linear in its first argument.

The above definition has the property that if  $c_f$  is a code for  $f$ , then

$$\frac{df(x, y)}{dx}(a) \cdot v = \frac{dc_f(x) y}{dx}(a) \cdot v = \left( \frac{dc_f(x)}{dx}(a) \cdot v \right) y$$

so that the derivative of  $f$  in it's first argument is given by the derivative of  $c_f$ .

A differential Turing category  $\mathbb{X}$  has **differential canonical codes** when  $\mathbb{X}$  has additive canonical codes  $\lambda(\_)$  that satisfy in addition:

$$D[\lambda(f)] = \lambda(\langle \pi_0 \times 0, \pi_1 \times 1 \rangle D[f])$$

The following is a recognition theorem for differential Turing categories.

**Proposition 4.8** *A Cartesian differential category  $\mathbb{X}$  is a differential Turing category if and only if*

- (i)  $\mathbb{X}$  has a differentially universal object  $T$ ;
- (ii) *There is a map  $T \times T \xrightarrow{\bullet} T$  that is universal for maps  $T \times T \rightarrow T$  and that is linear in its first argument.*

*Furthermore,  $\mathbb{X}$  is a differential Turing category with differential canonical codes if and only if (i) and (ii) hold and  $T \times T \xrightarrow{\bullet} T$  has differential canonical codes.*

**Proof.** The usual retraction is linear; this follows as  $T \times 1 \xrightarrow{\bullet} T$  is linear in its first argument, and ignores its second.

To show that we can construct differentially canonical codes for  $A \times B \rightarrow C$  from canonical codes for  $T \times T \rightarrow T$ , consider the diagram that codes  $\langle \pi_0 \times 0, \pi_1 \times 1 \rangle D[f]$  and use that  $r$  is linear, so that one takes the canonical code for  $(r \times r) \langle \pi_0 \times 0, \pi_1 \times 1 \rangle (s \times s) D[rfs]$ .  $\square$

For completeness we show that  $\mathcal{C}(\Lambda_{\beta\partial})$  is a differential Turing category with differentially canonical codes. We will first show that  $\mathcal{C}(\Lambda_{\beta\partial})$  is a Cartesian differential category in which  $U$  is a differentially universal object using proposition 4.7.

The differential structure on  $\mathcal{C}(\Lambda_{\beta\partial})$  is defined as follows:



$$\begin{aligned}
 D[p : S \vdash m : 1] &:= () \\
 D[p : S \vdash m : U] &:= (v, p) : S \times S \vdash \frac{dm}{dp}(p) \cdot v : U \\
 D[p : S \vdash (m_1, m_2) : R \times T] &:= \langle D[p : S \vdash m_1 : R], D[p : S \vdash m_2 : T] \rangle
 \end{aligned}$$

**Proposition 4.9**  $\mathcal{C}(\Lambda_{\beta\partial})$  is a Cartesian differential category in which  $U$  is a differentially universal object.

This proposition uses the fact that Curry’s retraction is linear, and that this is enough to determine a differential universal structure on  $U$ .

This leads to:

**Theorem 4.10**  $\mathcal{C}(\Lambda_{\beta\partial})$  is a differential Turing category with differentially canonical codes.

The proof involves a relatively straightforward sequence of calculations which show that codes are differentially canonical.

#### 4.4 Interpreting the differential $\lambda$ -calculus

Let  $\mathbb{X}$  be a differential Turing category with differential canonical codes. We will show that there is a functor  $\mathcal{C}(\Lambda_{\beta\partial}) \rightarrow \mathbb{X}$  that preserves all the differential structure and the canonical differential Turing structure. The functor  $\llbracket - \rrbracket : \mathcal{C}(\Lambda_{\beta\partial}) \rightarrow \mathbb{X}$  is defined in the same way as before for variables, applications, abstractions, sums, 0, and tuples of terms. For the differentials:

$$\llbracket q \vdash \frac{dm}{dp}(a) \cdot v \rrbracket := \langle \langle \llbracket q \vdash u \rrbracket, 0 \rangle, \langle \llbracket q \vdash a \rrbracket, 1 \rangle \rangle D[\llbracket (p, q) \vdash t \rrbracket]$$

To show that  $\llbracket - \rrbracket$  is a functor a crucial step is to establish:

**Lemma 4.11 (Substitution lemma)** For  $\llbracket - \rrbracket : \mathcal{C}(\Lambda_{\beta\partial}) \rightarrow \mathbb{X}$  where  $\mathbb{X}$  is a differential Turing category with differential canonical codes,

$$\llbracket p \vdash m[n/q] \rrbracket = \llbracket p \vdash n \rrbracket \llbracket q \vdash m \rrbracket$$

This then allows:

**Proposition 4.12** When  $\mathbb{X}$  is a differential Turing category with differentially canonical codes, then

$$\llbracket - \rrbracket : \mathcal{C}(\Lambda_{\beta\partial}) \rightarrow \mathbb{X}$$

is a Cartesian differential functor.

#### 4.5 Differential reflexive objects

To obtain the analog of the Scott-Koymans theorem we split the retractively linear idempotents in a differential Turing category with differential canonical codes to obtain a closed differential Turing category. As a first step, note:

**Lemma 4.13** *In any Cartesian differential category, the class  $\mathcal{E}$  of retractively linear idempotents is closed to identities and products. Furthermore, retractively linear idempotents are retractively additive.*

The above lemma ensures that  $\text{Split}_{\mathcal{E}}(\mathbb{X})$  is a Cartesian left additive category. The following proposition shows that if we split retractively linear idempotents, we can lift the differential structure from a Cartesian differential category to this idempotent splitting.

**Proposition 4.14** *Let  $\mathbb{X}$  be a Cartesian differential category, and  $\mathcal{E}$  the class of retractively linear idempotents. Then there is a unique differential structure on  $\text{Split}_{\mathcal{E}}(\mathbb{X})$  in which all  $e \in \mathcal{E}$  split with linear retraction and for which the inclusion  $\mathbb{X} \hookrightarrow \text{Split}_{\mathcal{E}}(\mathbb{X})$  is a Cartesian differential functor.*

**Proof.** From 3.9, we know that as  $\mathcal{E}$  is a product closed of retractively additive idempotents that contains the identities, that  $\text{Split}_{\mathcal{E}}(\mathbb{X})$  is a Cartesian left additive category.

In order for  $\mathbb{X} \hookrightarrow \text{Split}_{\mathcal{E}}(\mathbb{X})$  to be a Cartesian differential functor, the differential on maps between identities  $f : 1_A \rightarrow 1_B$  is forced to be the differential from  $\mathbb{X}$ .

If each idempotent is to split with a linear retraction, the derivative must satisfy  $(s_A \times s_A)D[r_A f s_B]r_B = D[f]$ .

This means the derivative for a  $e \xrightarrow{f} e'$  is:

$$D[f] := (e_A \times e_A)D[e_A f e_B]e_B = (e_A \times e_A)D[f]e_B = D[f]e_B$$

The details of the proof that this does indeed give a differential structure on  $\text{Split}_{\mathcal{E}}(\mathbb{X})$  are relatively straightforward; the proof will be somewhat similar to the proof of proposition 4.7.  $\square$

Next, we show that differential Turing structure lifts to the idempotent splitting.

**Proposition 4.15** *When  $\mathbb{X}$  is a differential Turing category (with Turing object  $T$ ), then so is  $\text{Split}_{\mathcal{E}}(\mathbb{X})$  where  $\mathcal{E}$  is the class of retractively linear idempotents.*

**Proof.** We have already seen that  $\text{Split}_{\mathcal{E}}(\mathbb{X})$  is both a Cartesian differential category and a Turing category with Turing object  $1_T$ .

Each idempotent  $e$  is also a retract of the Turing object  $1_T$ .

Hence, using Proposition 4.8, it suffices to show that the Turing morphism

$$1_T \times 1_T \xrightarrow{\bullet} 1_T$$

is linear in its first argument, which is immediate.  $\square$

Finally, we show that if  $\mathcal{E}$  is the collection of retractively linear idempotents in a differential Turing category with canonical codes, then  $\text{Split}_{\mathcal{E}}(\mathbb{X})$  is a Cartesian closed differential Turing category. We do not get for free that  $\text{Split}_{\mathcal{E}}(\mathbb{X})$  is still a Cartesian closed category: we must show that  $\mathcal{E}$  is closed to forming internal homs. Once we have established this, we will have that  $\text{Split}_{\mathcal{E}}(\mathbb{X})$  is a Cartesian closed

category that is also a differential Turing category. Moreover, it is then immediate that the coherence for *Cartesian closed differential categories*

$$D[\lambda(f)] = \lambda(\langle \pi_0 \times 0, \pi_1 \times 1 \rangle D[f])$$

holds because  $\lambda(f)$  in  $\text{Split}_{\mathcal{E}}(\mathbb{X})$  is the canonical code  $\lambda(f)$  from  $\mathbb{X}$ . Thus, the proof of the following theorem requires only that we show retractively linear maps are closed to forming internal homs.

We make use of lemma 4.3. Also, recall that if  $h$  is linear in its first argument, then

$$\langle \pi_0 \times 0, \pi_1 \times 1 \rangle D[h] = a_{\times}(\langle 1, 0 \rangle \times 1) D[h] = a_{\times}(1 \times \pi_1) h = (\pi_0 \times 1) h$$

A few long calculations provide the proof of:

**Theorem 4.16** *When  $\mathcal{E}$  is the collection of retractively linear idempotents in a differential Turing category with canonical codes, then  $\text{Split}_{\mathcal{E}}(\mathbb{X})$  is a Cartesian closed differential Turing category.*

This allows us to conclude the Scott-Koymans theorem for the differential  $\lambda$ -calculus.

**Corollary 4.17** *When  $\mathcal{E}$  is the class of retractively linear maps of a differential Turing category with canonical codes, the Turing object  $1_T$  is a reflexive object in  $\text{Split}_{\mathcal{E}}(\mathbb{X})$  and the retraction is  $1_T \rightarrow [1_T, 1_T]$  is a linear map.*

Thus, every model of the differential  $\lambda$ -calculus may be seen to arise as a differential reflexive object in a Cartesian closed differential category.

## References

- [1] Birkedal, L., “Developing Theories of Types via Computability and Realizability,” Ph.D. thesis, Carnegie Mellon University (1999).
- [2] Blute, R., J. Cockett and R. Seely, *Differential categories*, in: *Mathematical Structures in Computer Science*, 2006.
- [3] Blute, R., J. Cockett and R. Seely, *Cartesian Differential Categories*, *Theory and Application of Categories* **22** (2009), pp. 622–672.
- [4] Blute, R., J. Cockett and R. Seely, *Cartesian differential storage categories*, *Theory and Application of Categories* (2015).
- [5] Boudol, G., *The Lambda-Calculus with Multiplicities*, in: *CONCUR '93 Proceedings of the 4th International Conference on Concurrency Theory*, 1993.
- [6] Boudol, G., P. Curien and C. Lavatelli, *A semantics for lambda calculi with resource*, *Mathematical Structures in Computer Science* (1999).
- [7] Boudol, G. and C. Laneve, *Lambda-Calculus, Multiplicities, and the pi-Calculus*, Technical report, Institut National de Recherche en Informatique et en Automatique (1995).
- [8] Bucciarelli, A., T. Ehrhard and G. Manzonetto, *Categorical Models for Simply Typed Resource Calculi*, *Electronic Notes in Theoretical Computer Science* **265** (2010), pp. 213–230.
- [9] Cockett, J. and P. Hofstra, *Introduction to Turing Categories.*, *Annals of Pure and Applied Logic*. **156**. (2008.), pp. 183–209.
- [10] Ehrhard, T., *On Köthe Sequence Spaces and Linear Logic*, in: *Mathematical Structures in Computer Science*, 2002.

- [11] Ehrhard, T., *Finiteness spaces*, Mathematical Structures in Computer Science (2005).
- [12] Ehrhard, T. and L. Regnier, *The Differential Lambda Calculus*, Theoretical Computer Science **309** (2003), pp. 1–41.
- [13] Fiore, M., *Differential structure in models of multiplicative biadditive intuitionistic linear logic*, TLCA (2007).
- [14] Koymans, C., *Models of lambda calculus*, Information and Control (1982).
- [15] Lambek, J. and P. Scott, “An introduction to higher order categorical logic,” Cambridge University Press, 1986.
- [16] Longo, G. and E. Moggi, *A category theoretic characterization of functional completeness*, Theoretical Computer Science (1990).
- [17] Manzonetto, G., *What is a Categorical Model of the Differential and the Resource Lambda Calculi?*, Mathematical Structures in Computer Science **22** (2012), pp. 451–520.
- [18] Scott, D., *Relating theories of the  $\lambda$ -calculus*, Essays on Combinatory Logic, Lambda Calculus, and Formalism (1980).

# The shuffle quasimonad and modules with differentiation and integration

Marc Bagnol, Richard Blute

*Department of Mathematics and Statistics  
University of Ottawa  
Ottawa, Ontario CANADA*

J.R.B. Cockett

*Department of Computer Science  
University of Calgary  
Calgary, Alberta CANADA*

J.S. Lemay

*Department of Mathematics and Statistics  
University of Calgary  
Calgary, Alberta CANADA*

---

## Abstract

*Differential linear logic* and the corresponding categorical structure, *differential categories*, introduced the idea of differential structure associated to a (co)monad. Typically in settings such as algebraic geometry, one expresses differential structure for an algebra by having a module with a derivation, i.e. a map satisfying the Leibniz rule. In the monadic approach, we are able to continue to work with algebras and derivations, but the additional structure allows us to define other rules of the differential calculus for such modules; in particular one can define a monadic version of the chain rule as well as other basic identities.

In attempting to develop a similar theory of integral linear logic, we were led to consider the *shuffle multiplication*. This was shown by Guo and Keigher to be fundamental in the construction of the free *Rota-Baxter algebra*, the Rota-Baxter equation being the integral analogue of the Leibniz rule. This shuffle multiplication induces a *quasimonad* on the category of vector spaces. The notion of quasimonad, called *r-unital monad* by Wisbauer, is slightly weaker than that of monad, but is still sufficient to define a sensible notion of module with differentiation and integration.

In this paper, we demonstrate this quasimonad structure, show that its free modules have both differential and integral operators satisfying the Leibniz and Rota-Baxter rules and satisfy the fundamental theorems of calculus.

*Keywords:* Linear Logic, Differential Categories, Rota-Baxter Algebras

---

## 1 Introduction

The theory of *differential linear logic* as introduced by Ehrhard and Regnier [9,10] extended Girard's linear logic to include an inference rule which captured differentiation syntactically. The corresponding categorical structure, *differential categories* [1], extended the traditional notion of Seely category [26] to include a differential

*This paper is electronically published in  
Electronic Notes in Theoretical Computer Science  
URL: [www.elsevier.nl/locate/entcs](http://www.elsevier.nl/locate/entcs)*

combinator. The monads  $T$  that arise in models of linear logic<sup>1</sup> have the additional structure of a commutative, associative algebra associated to every object of the form  $TV$ . Such monads are called *algebra modalities*. Given an algebra modality, we require a map  $d: TV \rightarrow V \otimes TV$  satisfying naturality and as in the theory of Kähler differentials [18,2], we require that the combinator satisfies the Leibniz rule of differential calculus, viewing  $V \otimes TV$  as a right  $TV$ -module. But the monadic structure allows us to express other rules of calculus such as the chain rule. For details see Section 2.

The research in this paper began with an attempt to carry out a similar program for the integral calculus. The analogue of the Leibniz rule for integral calculus is the *Rota-Baxter equation*. While not as well-known as the Leibniz rule and the theory of derivations on an algebra, the equation has been an object of significant study since the construction of the free Rota-Baxter algebra by Guo and Keigher [16] and especially since this equation has been observed to be significant in renormalization of perturbative quantum field theory. See [8] for an overview. For the history of the subject, we refer the reader to the monograph by Guo [14]. An idea of the far-reaching application of this equation can be found by considering [7,8,12,13,15,16,29] as well as the webpage of Li Guo, which has a detailed bibliography.

The significance of the shuffle multiplication is clear from the Guo-Keigher construction of the free commutative Rota-Baxter algebra. This operation is naturally defined on the tensor algebra of a vector space, but surprisingly the algebraic structure so obtained does not yield a monad but only the slightly weaker notion of *quasimonad*, which we denote by  $\S$ . Quasimonads retain sufficient structure to describe the integral and differential structure we are interested in. In particular, one can define the notion of algebra modality with respect to a quasimonad and we show that the shuffle multiplication does give an algebra modality.

The notion for integral calculus corresponding to a module with differentiation does not seem to have been explored as far as we have been able to find. Given a commutative algebra  $A$ , we define a *module with integration* to be a right  $A$ -module  $M$  with a map  $P: M \rightarrow A$  satisfying a version of the Rota-Baxter equation. See Section 6 for details. We show that for the shuffle quasimonad, there is a canonical natural transformation  $P: V \otimes \S V \rightarrow \S V$ , making  $V \otimes \S V$  a module with integration, and a map  $d: \S V \rightarrow V \otimes \S V$  making  $V \otimes \S V$  a module with differentiation. These two maps together satisfy both the first and second fundamental theorems of calculus. We call such modules *FTC-modules*.

**Remark 1.1** The authors would like to thank NSERC for its generous support. The first author also received funding from the Fields Institute.

## 2 Codifferential categories and algebra modalities

We now define the basic structures related to the theory of (co)differential categories.

**Definition 2.1** An *algebra modality* on a symmetric monoidal category  $\mathcal{C}$  consists of a monad  $(T, \mu, \eta)$  on  $\mathcal{C}$ , and for each object  $C$  in  $\mathcal{C}$ , a pair of morphisms (note we

<sup>1</sup> Actually we are working in the dual setting to that of model of linear logic, where one has comonads and associated coalgebras.

are denoting the tensor unit by  $k$ )

$$m : T(C) \otimes T(C) \longrightarrow T(C), \quad e : k \longrightarrow T(C)$$

making  $T(C)$  a commutative algebra such that this family of associative algebra structures satisfies evident naturality conditions [1].

**Definition 2.2** An additive symmetric monoidal category with an algebra modality is a *codifferential category* if it is also equipped with a *deriving transform*<sup>2</sup>, i.e. a transformation, natural in  $C$

$$d_{T(C)} : T(C) \longrightarrow C \otimes T(C)$$

satisfying the following four equations<sup>3</sup>:

- (d1)  $e; d = 0$  (Derivative of a constant is 0.)
- (d2)  $m; d = (d \otimes 1); (1 \otimes m) + (1 \otimes d); c; (1 \otimes m)$  (where  $c$  is the appropriate symmetry) (Leibniz Rule)
- (d3)  $\eta; d = 1 \otimes e$  (Derivative of a linear function is constant.)
- (d4)  $\mu; d = d; d \otimes \mu; 1 \otimes m$  (Chain Rule)

**Remark 2.3** For us, an additive category is simply one enriched over abelian monoids. For the remainder of the paper, we will assume we are working over an additive category, although some of the definitions do not require it.

The fundamental example of a codifferential category is the category of (discrete) vector spaces and linear maps. The monad is given by the symmetric algebra construction and the deriving transform is the usual differentiation of polynomials. We refer to [1] for further details. A topological example is given by the category of *convenient vector spaces* and continuous linear maps, which forms a differential category [3].

### 3 Quasimonads

We give an exposition of the idea of weakening the definition of monad. We follow the presentation of Wisbauer [28] which is based in part on the work of Böhm [5]. This weaker notion will be more relevant in the study of the shuffle multiplication.

**Remark 3.1** We have chosen to use the term *quasimonad* for what Wisbauer calls an *r-unital monad*. We note that this is different than what Wisbauer and Böhm call a *weak monad*. It is also different than what Hoofman and Moerdijk call a *semimonad* [21].

We begin with the following preliminary definitions.

**Definition 3.2** • Let  $\mathcal{C}$  be a category, a pair  $(F, \mu)$  is a *functor with multiplication* if  $F : \mathcal{C} \rightarrow \mathcal{C}$  and  $\mu : F^2 \rightarrow F$  is a natural transformation with  $F\mu; \mu = \mu_F; \mu$ .

<sup>2</sup> We use the terminology of a *deriving transform* in both differential and codifferential categories.

<sup>3</sup> For simplicity, we write as if the monoidal structure is strict.

- A triple  $(F, \mu, \eta)$  is a *q-unital monad* if  $(F, \mu)$  is a functor with multiplication and  $\eta: id_{\mathcal{C}} \rightarrow F$  is a natural transformation, called the *quasi-unit*. (No equations required.)
- The quasi-unit is *regular* if  $\eta$  is equal to the composite:

$$id_{\mathcal{C}} \xrightarrow{\eta} F \xrightarrow{F\eta} F^2 \xrightarrow{\mu} F$$

- The multiplication  $\mu$  is *compatible* if  $\mu$  is equal to the composite

$$FF \xrightarrow{F\eta F} FFF \xrightarrow{\mu F} FF \xrightarrow{\mu} F$$

We can now define the notion of quasimonad as follows:

**Definition 3.3** A triple  $(F, \mu, \eta)$  is a *quasimonad* if it is a *q-unital monad* and:

- $\eta$  is regular.
- $\mu$  is compatible.

Just as a monad is always induced by an adjunction, quasimonads are always induced by a pairing of functors, defined as follows. Let  $\mathcal{C}$  and  $\mathcal{D}$  be categories and suppose we have a pair of functors, as follows

$$L: \mathcal{C} \longrightarrow \mathcal{D} \quad R: \mathcal{D} \longrightarrow \mathcal{C}$$

A *pairing* between  $L$  and  $R$  is a pair of maps, natural in both variables, of the form:

$$\alpha: Hom_{\mathcal{D}}(LA, B) \longrightarrow Hom_{\mathcal{C}}(A, RB) \quad \beta: Hom_{\mathcal{C}}(A, RB) \longrightarrow Hom_{\mathcal{D}}(LA, B)$$

Given such a pairing, as in the case of an adjunction, we get natural transformations:

$$\eta_A: A \longrightarrow LR(A) \quad \epsilon_B: RL(B) \longrightarrow B$$

We then define  $F: \mathcal{C} \rightarrow \mathcal{C}$  by  $F = L; R$ , and  $\mu =: F^2 \rightarrow F$  by  $\mu = \epsilon_L; R$ .

**Definition 3.4** A pairing is *regular* if

$$\alpha; \beta; \alpha = \alpha \quad \text{and} \quad \beta; \alpha; \beta = \beta$$

Given a *q-unital monad*  $(F, \eta, \mu)$ , one defines a category of  $F$ -algebras similarly to the case of monads and we get a pairing  $(\alpha_F, \beta_F)$  just as one obtains an adjunction in the case of a monad.

**Theorem 3.5 (Wisbauer [28])** *The following are equivalent:*

- $(F, \mu, \eta)$  is a *quasimonad*.
- The pairing  $(\alpha_F, \beta_F)$  is *regular*.

**Remark 3.6** In the case of a *q-unital monad*, the Kleisli construction yields an associative composition, but no identity maps. In the shuffle structure defined below, one in fact obtains a one-sided unit for the Kleisli construction.



## 4 Shuffling

We describe a quasimonad structure which will be fundamental in our definition and examples. We were led to consider this operation by the fundamental work of Guo and Keigher [15,16]. We work in the category of vector spaces over an arbitrary field  $k$ . So let

$$\S(V) = k \oplus V \oplus V \otimes V \oplus V \otimes V \otimes V \dots$$

This has a well-known monad structure as it is the free tensor algebra. But it also has a quasimonad structure which we describe now. We will work with homogeneous elements. See [24], Chapter 16.7. Denote the length of a homogeneous element  $w$  by  $|w|$ .

We have the evident free multiplication on  $\S(V)$ , but we also have the shuffle multiplication  $\diamond: \S(V) \otimes \S(V) \rightarrow \S(V)$  described as follows. We first remind the reader of the following preliminaries:

**Definition 4.1** • The generalized binomial coefficients are defined by

$$\binom{n_1 + n_2 \dots + n_m}{n_1, n_2 \dots, n_m} = \frac{(n_1 + n_2 \dots + n_m)!}{n_1! n_2! \dots n_m!}$$

with each  $n_i$  a non-negative integer. These coefficients satisfy evident equations which will be of use in verifying associativity of multiplication, among other things.

- If  $w_1$  and  $w_2$  are words in some alphabet, a *shuffle* of  $w_1$  and  $w_2$  is a permutation of the concatenated word  $w_1 w_2$  such that the internal order of the two words is maintained.

If  $w_1, w_2$  are homogeneous elements of  $\S V$ , then define

$$w_1 \diamond w_2 = \frac{1}{\binom{|w_1|+|w_2|}{|w_1|, |w_2|}} \sum_{w \in \text{Sh}(w_1, w_2)} w$$

Here the sum is over all  $w$  which are the shuffle of the two words. (We will find it convenient to denote the shuffle multiplication without the leading coefficient by  $w_1 * w_2$ .)

So for example, the product of  $w_1 = a_1 \otimes b_1$  and  $w_2 = a_2 \otimes b_2$  is

$$\begin{aligned} w_1 \diamond w_2 &= \frac{1}{6} [a_1 \otimes b_1 \otimes a_2 \otimes b_2 + a_1 \otimes a_2 \otimes b_1 \otimes b_2 + a_1 \otimes b_1 \otimes b_2 \otimes a_2 \\ &\quad + b_1 \otimes a_1 \otimes b_2 \otimes a_2 + b_1 \otimes a_1 \otimes a_2 \otimes b_2 + b_1 \otimes b_2 \otimes a_1 \otimes a_2] \\ &= \frac{1}{6} w_1 * w_2 \end{aligned}$$

We also note that the multiplication  $*$  can be defined recursively as follows [14]. If  $w_1 = aw'_1$  and  $w_2 = bw'_2$ , then:

$$w_1 * w_2 = a(w'_1 * w_2) + b(w_1 * w'_2)$$

Due to basic combinatorial identities of the binomial coefficients, the  $\diamond$ -operation is a commutative, unital associative multiplication on  $\S(V)$  and so induces a series of maps:

$$\S(V)^{\otimes n} \longrightarrow \S(V)$$

These maps can be defined directly via the formula:

$$w_1 \diamond w_2 \diamond \dots \diamond w_n = \frac{1}{\binom{|w_1|+|w_2|+\dots+|w_n|}{|w_1|, |w_2|, \dots, |w_n|}} \sum_{w \in \text{Sh}(w_1, w_2, \dots, w_n)} w$$

The multiplication  $\diamond$  induces a natural transformation  $\mu: \S\S \rightarrow \S$ . There is also an evident natural transformation  $\eta: Id \rightarrow \S$ , which is the usual *inclusion of generators* function.

**Theorem 4.2** *This makes  $\S$  a quasimonad.*

**Proof.** We prove the result in steps.

- $(\S, \mu)$  is a functor with multiplication.

We need to establish some notation for the homogeneous elements of the various iterates  $\S^n V$ :

- We write the elements of  $V$  as  $\{x_i\}_{i \in I}$ .
- We write the (homogeneous) elements of  $\S V$  as  $(x_1 x_2 \dots x_n)$ . In particular  $x_i \in V$  and  $(x_i) \in \S V$ . So  $(x_i)$  is the word of length one. We also have the empty word  $\varepsilon$  in all  $\S^n V$ , and note for example that  $(\varepsilon) \neq \varepsilon$  in  $\S^2(V)$ .
- We write the elements of  $\S^2 V$  as

$$[(x_{11} x_{12} \dots x_{1n_1})(x_{21} x_{22} \dots x_{2n_2}) \dots (x_{m1} x_{m2} \dots x_{mn_m})].$$

We will also write an element of this form as  $[w_1 w_2 \dots w_m]$ .

- We write the elements of  $\S^3 V$  as  $[w_{11} w_{12} \dots w_{1m_1}] \dots [w_{p1} w_{p2} \dots w_{pm_p}]$

Now calculate as follows:

$$\begin{aligned} & \S \mu([w_{11} w_{12} \dots w_{1m_1}] \dots [w_{p1} w_{p2} \dots w_{pm_p}]) = \\ & [w_{11} \diamond w_{12} \diamond \dots \diamond w_{1m_1}] \dots [w_{p1} \diamond w_{p2} \diamond \dots \diamond w_{pm_p}] = \\ & \frac{1}{\binom{|w_{11}|+|w_{12}|+\dots+|w_{1m_1}|}{|w_{11}|, |w_{12}|, \dots, |w_{1m_1}|}} \dots \frac{1}{\binom{|w_{p1}|+|w_{p2}|+\dots+|w_{pm_p}|}{|w_{p1}|, |w_{p2}|, \dots, |w_{pm_p}|}} [w_{11} * w_{12} * \dots \\ & \quad * w_{1m_1}] \dots [w_{p1} * w_{p2} * \dots * w_{pm_p}] \end{aligned}$$

Applying  $\mu$  to this element and using combinatorial identities, we get:

$$\begin{aligned}
 & \frac{1}{\binom{|w_{11}|+|w_{12}|+\dots+|w_{1m_1}|+\dots+|w_{p1}|+|w_{p2}|+\dots+|w_{pm_p}|}{|w_{11}|, |w_{12}|, \dots, |w_{1m_1}|, \dots, |w_{p1}|, |w_{p2}|, \dots, |w_{pm_p}|}} w_{11} * w_{12} \dots * w_{1m_1} * \dots \\
 & * w_{p1} * w_{p2} \dots * w_{pm_p} = \\
 & w_{11} \diamond w_{12} \diamond \dots \diamond w_{1m_1} \diamond \dots \diamond w_{p1} \diamond w_{p2} \diamond \dots \diamond w_{pm_p}
 \end{aligned}$$

On the other hand, we have

$$\begin{aligned}
 & \mu_{\S}([w_{11}w_{12} \dots w_{1m_1}] \dots [w_{p1}w_{p2} \dots w_{pm_p}]) = \\
 & [w_{11}w_{12} \dots w_{1m_1}] \diamond \dots \diamond [w_{p1}w_{p2} \dots w_{pm_p}] = \\
 & \frac{1}{\binom{m_1+\dots+m_p}{m_1, m_2, \dots, m_p}} [w_{11}w_{12} \dots w_{1m_1}] * \dots * [w_{p1}w_{p2} \dots w_{pm_p}]
 \end{aligned}$$

Note that in this multiplication we are viewing the  $w$ 's as letters. Now note  $\mu([w_{11}w_{12} \dots w_{1m_1}] * \dots * [w_{p1}w_{p2} \dots w_{pm_p}])$  is:

$$\binom{m_1 + \dots + m_p}{m_1, m_2, \dots, m_p} [w_{11} \diamond w_{12} \diamond \dots \diamond w_{1m_1} \diamond \dots \diamond w_{p1} \diamond w_{p2} \diamond \dots \diamond w_{pm_p}]$$

since we have  $\binom{m_1+\dots+m_p}{m_1, m_2, \dots, m_p}$  terms in the product  $[w_{11}w_{12} \dots w_{1m_1}] * \dots * [w_{p1}w_{p2} \dots w_{pm_p}]$  each of which gives  $w_{11} \diamond w_{12} \diamond \dots \diamond w_{1m_1} \diamond \dots \diamond w_{p1} \diamond w_{p2} \diamond \dots \diamond w_{pm_p}$  when we apply  $\mu$ .

Thus we have a functor with multiplication.

- $\eta$  is regular.

We note that  $\S\eta$  is just  $id \oplus \eta \oplus (\eta \otimes \eta) \oplus \dots$ . So  $\eta; \S\eta$  is just the map  $v \mapsto (v)$ , viewing  $v$  as a word of length 1. Then  $\mu((v)) = v$ .

- $\mu$  is compatible.

We consider a typical element of  $\S^2 V$  given by:

$$[(x_{11}x_{12} \dots x_{1n_1})(x_{21}x_{22} \dots x_{2n_2}) \dots (x_{m1}x_{m2} \dots x_{mn_2})]$$

We also denote this by  $[w_1w_2 \dots w_m]$ . The action of the map  $F\eta_F$  on this element is to send it to  $[w_1][w_2] \dots [w_m]$ , where each  $(w_i)$  is a word of length one in  $\S^2 V$ . Applying  $\mu_{\S}$  to this element gives:

$$\frac{1}{m!} [\Sigma_m([(w_1)(w_2) \dots (w_m)])]$$

where  $\Sigma_m$  indicates the sum over the action of the permutation group  $S_m$  on the list  $[(w_1)(w_2) \dots (w_m)]$ . Applying  $\mu$  to  $[\Sigma_m([(w_1)(w_2) \dots (w_m)])]$ , we get  $m!$  copies of  $w_1 \diamond w_2 \diamond \dots \diamond w_m$ , and we are done.

This completes the proof that  $(\S, \mu, \eta)$  is a quasimonad. □

**Definition 4.3** A  $q$ -unital monad or quasimonad  $\S$  is an *algebra modality* if for each object  $V$ , there is an associative algebra structure:

$$\diamond: \S V \otimes \S V \rightarrow \S V \quad e: I \rightarrow \S V$$

which is natural in  $V$  and the following two additional equations hold:

$$\begin{array}{ccc} \S \S V \otimes \S \S V & \xrightarrow{\mu \otimes \mu} & \S V \otimes \S V \\ \diamond \downarrow & & \downarrow \diamond \\ \S \S V & \xrightarrow{\mu} & \S V \end{array}$$

$$\begin{array}{ccc} I & \xrightarrow{e} & \S \S V \\ & \searrow e & \downarrow \mu \\ & & \S V \end{array}$$

These equations say the  $\mu$  is an algebra homomorphism.

**Lemma 4.4** *The shuffle multiplication makes  $\S$  an algebra modality.*

**Proof.** The second equation is straightforward. For the first equation, we proceed very much as in the case of the proof that we have a functor with multiplication. So we consider an expression of the form  $[w_1 w_2 \dots w_m] \otimes [u_1 u_2 \dots u_n]$ . Applying  $\mu \otimes \mu$  and then  $\diamond$ , we get

$$\begin{aligned} [w_1 w_2 \dots w_m] \otimes [u_1 u_2 \dots u_n] &\mapsto [w_1 \diamond w_2 \dots \diamond w_m] \otimes [u_1 \diamond u_2 \dots \diamond u_n] \\ &\mapsto w_1 \diamond w_2 \diamond \dots \diamond w_m \diamond u_1 \diamond u_2 \diamond \dots \diamond u_n \end{aligned}$$

Applying  $\diamond$  then  $\mu$  gives

$$\begin{aligned} [w_1 w_2 \dots w_m] \otimes [u_1 u_2 \dots u_n] &\mapsto \frac{1}{\binom{m+n}{m,n}} (w_1 w_2 \dots w_m) * (u_1 u_2 \dots u_n) \mapsto \\ &\frac{1}{\binom{m+n}{m,n}} \binom{m+n}{m,n} w_1 \diamond w_2 \diamond \dots \diamond w_m \diamond u_1 \diamond u_2 \diamond \dots \diamond u_n \end{aligned}$$

and the result follows. □

## 5 The Rota-Baxter equation

We now introduce the Rota-Baxter equation and give examples. All of the material of this section can be found in [14].

**Definition 5.1** Let  $A$  be a  $k$ -algebra, where  $k$  is the underlying field.  $A$  is a *Rota-Baxter algebra* if equipped with a  $k$ -linear map  $P: A \rightarrow A$  such that for all  $x, y \in A$

$$P(x)P(y) = P(xP(y)) + P(P(x)y)$$

The map  $P$  is called a *Rota-Baxter operator* or *RB-operator*<sup>4</sup>.

We just mention a few examples. A much more extensive list can be found for example in [14].

- Let  $\mathcal{C}(\mathbb{R})$  denote the ring of continuous functions from the reals to the reals under pointwise operations. Define  $P(f)(x) = \int_0^x f(t)dt$ . Then  $P$  is an RB-operator. The Rota-Baxter equation becomes the usual integration by parts formula.
- Consider  $\mathbb{R}[x]$  with multiplication given by  $x^m \cdot x^n = \binom{m}{n}x^{m+n}$ . Then  $P(x^n) = x^{n+1}$  is an RB-operator.
- Let  $V$  be an arbitrary  $k$ -vector space. Let  $T(V) = k \oplus V \oplus V \otimes V \dots$ , but equipped with the shuffle algebra multiplication. Then if  $v \in V$ , we have an operator  $P_v: T(V) \rightarrow T(V)$  defined by  $P_v(w) = v \otimes w$ . Then  $P_v$  is an RB-operator.

## 6 Modules with differentiation and integration

The notion of derivation has long been fundamental in algebraic geometry and commutative algebra [18,25] and more recently extending the idea to the noncommutative setting has also been of importance [23]. We begin with the classical notion:

**Definition 6.1** Let  $A$  be a commutative  $k$ -algebra. Let  $M$  be a (left)  $A$ -module. A *derivation* on  $M$  is a  $k$ -linear map  $\partial: A \rightarrow M$  such that for all  $x, y \in A$

$$\partial(xy) = x\partial(y) + y\partial(x)$$

We will also refer to  $(M, \partial)$  as a *module with differentiation*.

We now introduce the corresponding integral structure. As far as we have been able to see, this precise definition does not exist in the literature despite the intense study of the Rota-Baxter equation. It is certainly implicit in that work though.

**Definition 6.2** Let  $A$  be a commutative  $k$ -algebra. Let  $M$  be a right  $A$ -module. An *integration* on  $M$  is a  $k$ -linear map  $\pi: M \rightarrow A$  such that for all  $x, y \in M$

$$\pi(x)\pi(y) = \pi(x\pi(y)) + \pi(y\pi(x))$$

The pair  $(M, \pi)$  is called a *module with integration*.

**Remark 6.3** Note that the multiplication on the lefthand side of the equation is the multiplication of  $A$ , while on the right, the multiplication is the action of  $A$  on  $M$ .

<sup>4</sup> In this paper, we only consider the operators of weight 0.

We note that every Rota-Baxter algebra is a module with integration over itself with its evident right-module structure. The shuffle quasimonad will give us a much broader class of examples. Indeed it is expected that when a complete theory of integral linear logic is established, we will have an even greater source of examples.

**Definition 6.4** Let  $A$  be a commutative algebra. An *FTC-module over  $A$*  is an  $A$ -module  $M$  together with maps  $P: M \rightarrow A$  and  $d: A \rightarrow M$  such that

- $(M, d)$  is a module with differentiation.
- $(M, P)$  is a module with integration.
- and
- (First Fundamental Theorem of Calculus)  $P; d = id$

We write the FTC-module as  $(M, P, d)$ .

### 6.1 Additional structure in the presence of a (quasi)monad

As already indicated, one can express additional differential structure in the presence of a monad with an algebra modality. This is seen in the definition of codifferential category above. We now introduce some additional structure for the integral case.

**Definition 6.5** In what follows, let  $(T, \mu, \eta)$  be a  $q$ -unital monad and a natural transformation of the form  $s: id \otimes T \rightarrow T$

- The natural transformation  $s$  satisfies the *U-substitution rule* if for all  $f: X \rightarrow X \otimes TX$ , the composite

$$X \otimes TX \xrightarrow{s} TX \xrightarrow{Tf} T(X \otimes TX) \xrightarrow{Ts} TTX \xrightarrow{\mu} TX$$

is equal to the composite

$$\begin{aligned} X \otimes TX &\xrightarrow{id \otimes Tf} X \otimes T(X \otimes TX) \xrightarrow{id \otimes Ts} X \otimes TTX \\ &\xrightarrow{f \otimes \mu} X \otimes TX \otimes TX \xrightarrow{id \otimes \diamond} X \otimes TX \xrightarrow{s} TX \end{aligned}$$

- A natural transformation of the form  $s: id \otimes T \rightarrow T$  satisfies the *integration of constants* rule if  $\eta: X \rightarrow TX$  is equal to the composite

$$X \cong X \otimes I \xrightarrow{id \otimes e} X \otimes TX \xrightarrow{s} TX$$

We note that these equations are not necessarily satisfied in the case of the shuffle quasimonad and it will be of interest to characterize those cases in which these additional equations hold.

If we also have differential structure in the presence of a quasimonad, we can also state the *Second Fundamental Theorem of Calculus*.

**Definition 6.6** Suppose we have an algebra modality  $(T, \mu, \eta, m, e)$  and an FTC-module  $(M, P, d)$  over  $T(V)$ . Then we say that  $M$  satisfies the *Second Fundamental Theorem of Calculus* if:

$$d; P + T(0) = id_{T(V)}$$

where  $0: V \rightarrow V$ .

If the algebra modality is equipped with natural transformations  $P: id \otimes T \rightarrow T$  and  $d: T \rightarrow id \otimes T$  making each  $TV$  an FTC-module, then we say that *the algebra modality satisfies the second fundamental theorem* if these natural transformations satisfy the same equation.

**Remark 6.7** We note that unlike the first fundamental theorem of calculus, this one can only be defined in the presence of additional quasimonadic structure.

## 7 Differential and integral structure in the shuffle quasimonad

**Lemma 7.1** *The operator  $P: V \otimes \S V \rightarrow \S V$  defined by  $P(v \otimes w) = \frac{1}{|w|+1}vw$  (the concatenated word) satisfies:*

- *The Rota-Baxter equation, where  $V \otimes \S V$  is the free right  $\S V$ -module generated by  $V$ .*
- *The integration of constants rule.*

**Proof.** We note that the integration of constants rule is trivial.

We suppose  $v, v' \in V$  and  $w, w' \in X^*$ , with  $|w| = n$  and  $|w'| = m$ . We must show

$$P(v \otimes w)P(v' \otimes w') = P((v \otimes w)P(v' \otimes w')) + P((v' \otimes w')P(v \otimes w))$$

The lefthand side of this equation is given by:

$$\begin{aligned} \frac{1}{n+1} \frac{1}{m+1} [vw \diamond v'w'] &= \frac{1}{n+1} \frac{1}{m+1} \frac{1}{\binom{n+m+2}{n+1, m+1}} (vw * v'w') = \\ &\frac{1}{\binom{n+m+2}{n, 1, m, 1}} (vw * v'w') \end{aligned}$$

The righthand side is given by:

$$\begin{aligned} &P(v \otimes (\frac{1}{m+1})w \diamond v'w') + P(v' \otimes (\frac{1}{n+1})w' \diamond vw) = \\ &\frac{1}{m+1} \frac{1}{\binom{n+m+1}{n, m+1}} \frac{1}{n+m+2} v(w * v'w') + \frac{1}{n+1} \frac{1}{\binom{n+m+1}{n+1, m}} \frac{1}{n+m+2} v'(w' * vw) = \\ &\frac{1}{\binom{n+m+2}{n, 1, m, 1}} v(w * v'w') + \frac{1}{\binom{n+m+2}{n, 1, m, 1}} v'(w' * vw) \end{aligned}$$

The result now follows from the recursive definition of the  $*$ -operator. □

**Lemma 7.2** *In the category of vector spaces equipped with the quasishuffle algebra modality, for each algebra  $\S V$ , the differential operator given by:*

$$d: \S V \rightarrow V \otimes \S V \quad vw \mapsto (|w| + 1)v \otimes w$$

*satisfies the Leibniz rule.*

**Proof.** We must show that

$$d(vw \diamond v'w') = d(vw) \diamond v'w' + d(v'w') \diamond vw$$

Note we are using the  $\diamond$  operation to also signify the action of  $\S V$  on  $V \otimes \S V$ . We let  $|w| = n$  and  $|w'| = m$ .

For the lefthand side, we calculate:

$$\begin{aligned} d(vw \diamond v'w') &= d\left[\frac{1}{\binom{n+m+2}{n+1, m+1}}(v(v'w' * w) + v'(vw * w'))\right] = \\ &= \frac{1}{\binom{n+m+2}{n+1, m+1}}[(n+m+2)(v \otimes (w * v'w') + v' \otimes (vw * w'))] = \\ &= \frac{(n+1)!(m+1)!}{(n+m+1)!}[v \otimes (w * v'w') + v' \otimes (vw * w')] \end{aligned}$$

For the righthand side, we calculate:

$$\begin{aligned} d(vw) \diamond v'w' + d(v'w') \diamond vw &= (n+1)v \otimes (w \diamond v'w') + (m+1)v' \otimes (w' \diamond vw) = \\ &= (n+1)\frac{1}{\binom{n+m+1}{n, m+1}}v \otimes (w * v'w') + (m+1)\frac{1}{\binom{n+m+1}{n+1, m}}v' \otimes (w' * vw) = \\ &= \frac{(n+1)!(m+1)!}{(n+m+1)!}[v \otimes (w * v'w') + v' \otimes (vw * w')] \end{aligned}$$

□

Finally we conclude:

**Theorem 7.3** *For the algebra modality  $\S$ , the free  $\S V$  module on  $V$  given by  $V \otimes \S V$  is an FTC-module which furthermore satisfies the second fundamental theorem of calculus.*

**Proof.** It remains to verify the two fundamental theorems. The first is straightforward.

We must consider the two cases of monomials  $V^{\otimes n}$ , when  $n = 0$  and  $n \geq 1$ . When  $n = 0$ ,  $k \in \mathbb{K}$ , recall that  $d(1) = 0$  and  $P$  is linear:

$$P(d(k)) + \S(0)(k) = P(0) + k = k$$

When  $n \geq 1$ , then for  $vw \in V^{\otimes n}$  (where  $v \in V$  and  $w \in \S(V)$  of length  $|w| = n - 1$ ):

$$P(d(vw)) + \S(0)(vw) = P((|w| + 1)v \otimes w) = \frac{(|w| + 1)}{(|w| + 1)}vw = vw$$



This establishes the second FTC. □

We now consider the possibility of other  $\S V$  modules satisfying the second fundamental theorem. We will show that requiring the second fundamental theorem is in fact a significant restriction.

**Lemma 7.4** *Let  $(M, P, d)$  be an FTC-module over  $\S V$  which satisfies the Second Fundamental Theorem of calculus. Then the following equality holds:*

$$P; \S(0) = 0$$

**Proof.**

$$\begin{aligned} P; \S(0) &= P + P; \S(0) - P \\ &= P; d; P + P; \S(0) - P \\ &= P(d; P + \S(0)) - P \\ &= P - P \\ &= 0 \end{aligned}$$

□

**Proposition 7.5** *For the algebra  $\S V$ , we consider the FTC-module  $(V \otimes \S V, P, d)$  as above. Suppose I also have another FTC-module  $(M, R, d)$  over  $\S V$  which also satisfies the second fundamental theorem. Then there is a  $k$ -linear isomorphism between  $M$  and  $V \otimes \S V$  given by*

$$P; D: V \otimes \S V \longrightarrow M \quad R; d: M \longrightarrow V \otimes \S V$$

Furthermore, if  $P; D$  satisfies the following for all  $a \otimes w \in V \otimes \S V$ :

$$D(P(a \otimes w)) = wD(a)$$

then  $P; D$  is a module map, implying  $V \otimes \S V$  and  $M$  are isomorphic as  $\S V$ -modules.

**Proof.**

By the above lemma,  $R; \S(0) = 0$  and  $P; \S(0) = 0$ , and so we get the following equalities:

$$R = R(dP + \S(0)) = RdP + R\S(0) = RdP \quad P = P(DR + \S(0)) = PDR + P\S(0) = PDR$$

So calculate as follows:

$$P; D; R; d = P; d = id_{V \otimes \S V}$$

$$R; d; P; D = R; D = id_M$$

So  $PD$  and  $Rd$  are  $\mathbb{K}$ -linear isomorphisms.

Now suppose that for all  $a \otimes w \in V \otimes \S V$ :  $D(P(a \otimes w)) = wD(a)$ . By a simple calculation we have that for all  $v \in \S(V)$ :

$$\begin{aligned} D(P(v(a \otimes w))) &= D(P(a \otimes v \diamond w)) \\ &= (v \diamond w)D(a) \\ &= v(wD(a)) \\ &= v(D(P(a \otimes w))) \end{aligned}$$

Which proves that  $P; D$  is a module map. □

## 8 Conclusion

This work originated with the goal of developing a theory of integral linear logic and integral categories to parallel the corresponding differential theories. This work is ongoing but we believe the shuffle structure provides a key towards understanding the integral theory. But furthermore it is of interest even in its own right. The idea of weakening the notion of monad to quasimonad is new for linear logic and deserves further exploration. (We do note that a different version of weaker structure was introduced in [20,21].) Also, we find the combinatorics of shuffling and its variants fascinating and wonder what other structure is to be found there and what it would have to say about linear logic.

We also note that one can still consider  $T$ -algebras when  $T$  is just a quasimonad. See [28]. An extension of the theory of universal derivations established in [2] for general  $T$ -algebras was carried out by O’Neill in [27]. This theory was subsequently subsumed in [4] where the general notion of a  $T$ -derivation with respect to an algebra modality was introduced. It will be interesting to see the extent to which the work there lifts to the quasimonad setting.

The notion of Rota-Baxter algebra as studied in [14] and the references therein is in fact much more general than the definition presented here. In particular, they have the notion of *Rota-Baxter algebra of weight  $\lambda$* . The definition is as follows:

**Definition 8.1** Let  $A$  be a  $k$ -algebra.  $A$  is a *Rota-Baxter algebra of weight  $\lambda$*  if equipped with a  $k$ -linear map  $P: A \rightarrow A$  such that for all  $x, y \in A$

$$P(x)P(y) = P(xP(y)) + P(P(x)y) + \lambda P(xy)$$

Our notion of module with integration only captures the weight 0 case. But there is an evident notion of module with integration of weight  $\lambda$ . The logical significance of this is likely quite interesting. At the same time, Guo and Keigher have also developed a corresponding notion of *differential algebra of weight  $\lambda$* , defined as follows:

**Definition 8.2** Let  $A$  be a  $k$ -algebra.  $A$  is a *differential algebra of weight  $\lambda$*  if equipped with a  $k$ -linear map  $d: A \rightarrow A$  such that for all  $x, y \in A$

$$d(xy) = xd(y) + d(x)y + \lambda d(x)d(y)$$

They combine the two structures in [16]. The paper [17] studies the corresponding monadic and comonadic structures. Obviously there is a great deal of structure here to be studied.

We also note that there is a corresponding theory of *Rota-Baxter coalgebras* [22]. So many of the structures defined here could be redefined in the coalgebraic/comonadic setting. Of course, it remains to find as compelling an example as the shuffle structures considered here.

Two further ideas for future work are as follows. First it is important to develop the above theories in the noncommutative case. This work for the differential setting was begun in the preprint [6]. Free Rota-Baxter algebras in the noncommutative case are constructed by Ebrahimi-Fard and Guo using operations on rooted trees in [7]. It is this construction that arises in renormalization of perturbative quantum field theory [8].

We would also like to construct free FTC-modules in both the weight 0 and weight  $\lambda$  cases. Obviously these will be related to the structures found in [16,17]. This generalized notion of shuffle is also related to the *quasishuffle* of Hoffman [19].

## References

- [1] R. Blute, J.R.B. Cockett, R.A.G. Seely. Differential categories. *Mathematical Structures in Computer Science* 16, pp. 1049-1083, (2006).
- [2] R. Blute, R. Cockett, T. Porter, R. Seely. Kähler categories. *Cahiers de Topologie et Geometrie Differentielle* 52, pp. 253-268, (2011).
- [3] R. Blute, T. Ehrhard and C. Tasson. A Convenient Differential Category. *Cahiers de Topologie et Geometrie Differentielle* 53. pp. 211-233, (2012).
- [4] R. Blute, R.B.B. Lucyshyn-Wright, K. O'Neill. Derivations in codifferential categories. *to appear in Cahiers de Topologie et Geometrie Differentielle*, (2016).
- [5] G. Böhm. The weak theory of monads. *Adv. Math.* 225 pp. 1-32, (2010).
- [6] R. Cockett. Lectures on noncommutative Kähler categories. Preprint (2014).
- [7] K. Ebrahimi-Fard, L. Guo. Free Rota-Baxter algebras and rooted trees. *J. Algebra and Its Applications* 7, pp. 167-194 (2008).
- [8] K. Ebrahimi-Fard, L. Guo. Rota-Baxter Algebras in Renormalization of Perturbative Quantum Field Theory. *in Universality and Renormalization*, edited by I. Binder and D. Kreimer, (2007).
- [9] T. Ehrhard, L. Regnier *The differential  $\lambda$ -calculus*. *Theoretical Computer Science*, 309(1-3) (2003) 1–41.
- [10] T. Ehrhard, L. Regnier *Differential interaction nets*. Workshop on Logic, Language, Information and Computation (WoLLIC), invited paper. *Electronic Notes in Theoretical Computer Science*, vol. 123, March 2005, Elsevier.
- [11] J.-Y. Girard *Linear logic*. *Theoretical Computer Science* 50 (1987) 1–102.
- [12] L. Guo. Properties of free Baxter algebras, *Adv. Math.* 151, pp. 346– 374, (2000).
- [13] L. Guo. Baxter algebra and differential algebra, in *Differential Algebra and Related Topics*, World Scientific Publishing Company, 2002.
- [14] L. Guo. *An Introduction to Rota-Baxter algebra*. *Surveys of Modern Mathematics* 4, (2012).
- [15] L. Guo, W. Keigher. Baxter algebras and shuffle algebras. *Advances in Mathematics* 150, pp. 117-149, (2000).

- [16] L. Guo, W. Keigher. On differential Rota-Baxter algebras, J. Pure Appl. Algebra 212, pp. 522-540, (2008).
- [17] L. Guo, W. Keigher, S. Zhang. Monads and distributive laws for Rota-Baxter and differential algebras, preprint, (2014).
- [18] R. Hartshorne, *Algebraic Geometry*. Springer-Verlag, (1977).
- [19] M.E. Hoffman. Quasi-shuffle products. J. Algebraic Combinatorics , pp. 49-68, (2000).
- [20] R. Hoofman. Non-stable models of linear logic. *Logical Foundations of Computer Science*, Lecture Notes in Computer Science 605, pp. 209-220, (2005).
- [21] R. Hoofman, I. Moerdijk. A remark on the theory of semifunctors. Mathematical Structures in Computer Science 5, pp. 1-8, (1995).
- [22] R. Jian, J. Zhang. Rota-Baxter coalgebras, (preprint), (2014).
- [23] G. Landi. *An introduction to noncommutative spaces and their geometries*. Lecture Notes in Physics, Springer-Verlag, (1997).
- [24] S. Lang. *Algebra, Third Edition*, Springer Graduate Texts in Mathematics, Springer-Verlag, (2005).
- [25] H. Matsumura. *Commutative Ring Theory*, Cambridge University Press, (1986).
- [26] P.-A. Mellies. *Categorical semantics of linear logic*. Panoramas et Syntheses 27, Societe Mathematique de France, (2009).
- [27] T. O'Neill. *Differential Forms for T-Algebras in Kähler Categories*. M.Sc. Thesis, (2013).
- [28] R. Wisbauer. Regular pairings of functors and weak (co)monads. Algebra & Discrete Math 15, pp. 127-154, (2013).
- [29] S. Zheng, L. Guo, M. Rosenkranz. Rota-Baxter operators on the polynomial algebra, integration and averaging operators. Pacific Journal of Mathematics 275, pp. 481-506, (2015).